**ASSIGNMENT: Multiple Classes With Association**



Session: 2021 – 2024

**Submitted by:**

Muhammad Abubakar Siddique Farooqi 2021-CS-171

**Supervised by:**

Mam Maida Shahid

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**Table of Contents**

**Content**............................................................................................................................. **Page No**

Domain Model with Multiplicity..............................................................................................6

Domain Model With Relations And Constraints.....................................................................5

Coffee Shop..................................................................................................................................... 4

Case Study ................................................................................................................................. 4

Domain Model With Names......................................................................................................5

Class Diagram..............................................................................................................................6

Complete Code.............................................................................................................................7

Sequence Diagrams,.................................................................................................................17

Game Object....................................................................................................................................18

Sequence Diagrams,...................................................................................................................31

Complete Code.............................................................................................................................22

Class Diagram..............................................................................................................................21

Domain Model with Multiplicity..............................................................................................21

Domain Model With Relations And Constraints.....................................................................20

Domain Model With Names......................................................................................................20

Case Study ................................................................................................................................. 18

Ocean navigation.............................................................................................................................32

Case Study ................................................................................................................................. 32

Domain Model With Names......................................................................................................33

Sequence Diagrams,...................................................................................................................42

Complete Code.............................................................................................................................35

Class Diagram..............................................................................................................................34

Domain Model with Multiplicity..............................................................................................34

Domain Model With Relations And Constraints.....................................................................33

Point of Sale.....................................................................................................................................43

Case Study ................................................................................................................................. 43

Domain Model With Names......................................................................................................44

Domain Model With Relations And Constraints.....................................................................44

Domain Model with Multiplicity..............................................................................................45

Class Diagram..............................................................................................................................45

Complete Code.............................................................................................................................46

Sequence Diagrams,...................................................................................................................59

Point on Line....................................................................................................................................60

Case Study ................................................................................................................................. 60

Domain Model With Names......................................................................................................61

Domain Model With Relations And Constraints.....................................................................61

Domain Model with Multiplicity..............................................................................................62

Class Diagram..............................................................................................................................62

Complete Code.............................................................................................................................63

Sequence Diagrams,...................................................................................................................68

UAMS...............................................................................................................................................69

Case Study ................................................................................................................................. 69

Domain Model With Names......................................................................................................70

Sequence Diagrams,...................................................................................................................91

Complete Code.............................................................................................................................72

Class Diagram..............................................................................................................................71

Domain Model with Multiplicity..............................................................................................71

Domain Model With Relations And Constraints.....................................................................70

**Coffee Shop**

**Case Study**

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

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.

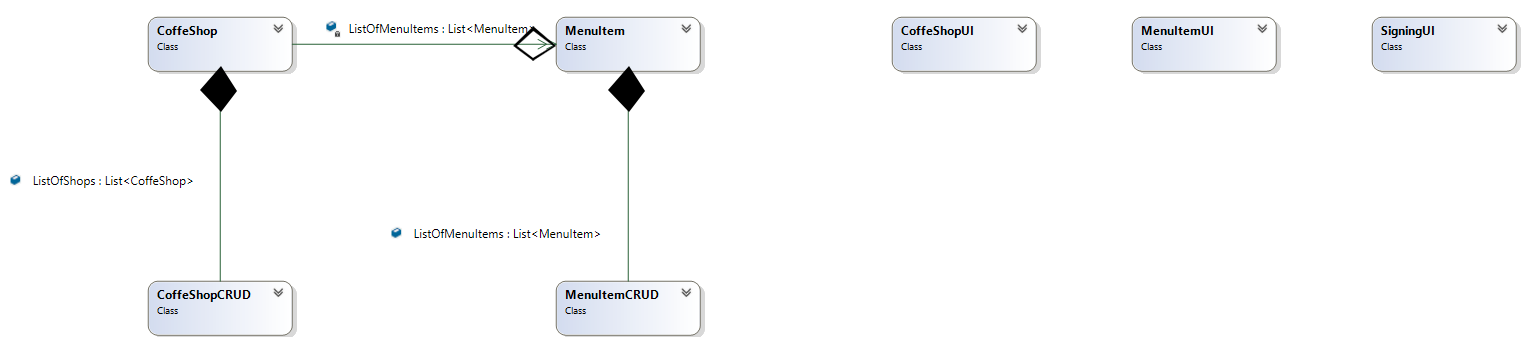
7. **drinksOnly**: returns only the item names of type drink from the menu.

8. **foodOnly**: returns only the item names of type food from the menu.

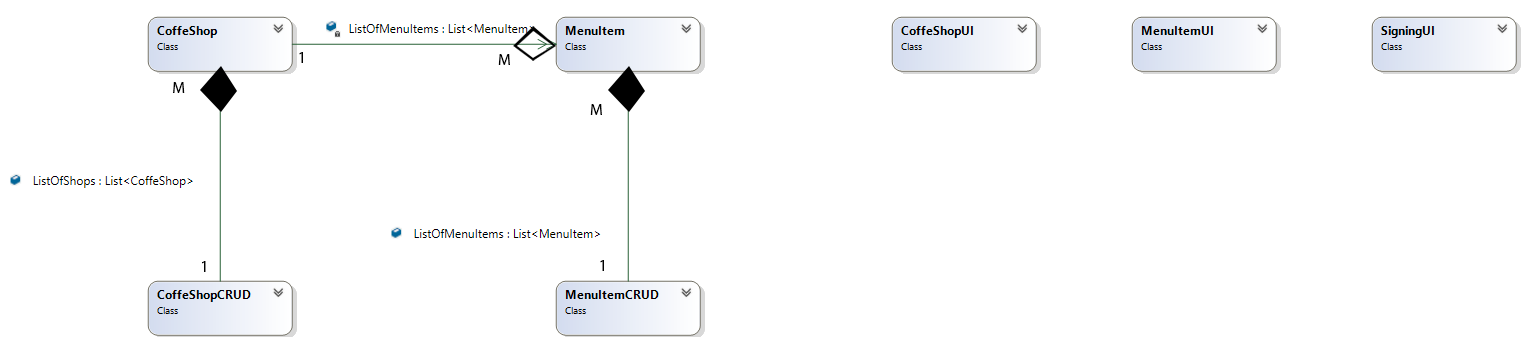
**Domain Model with only the Class names**



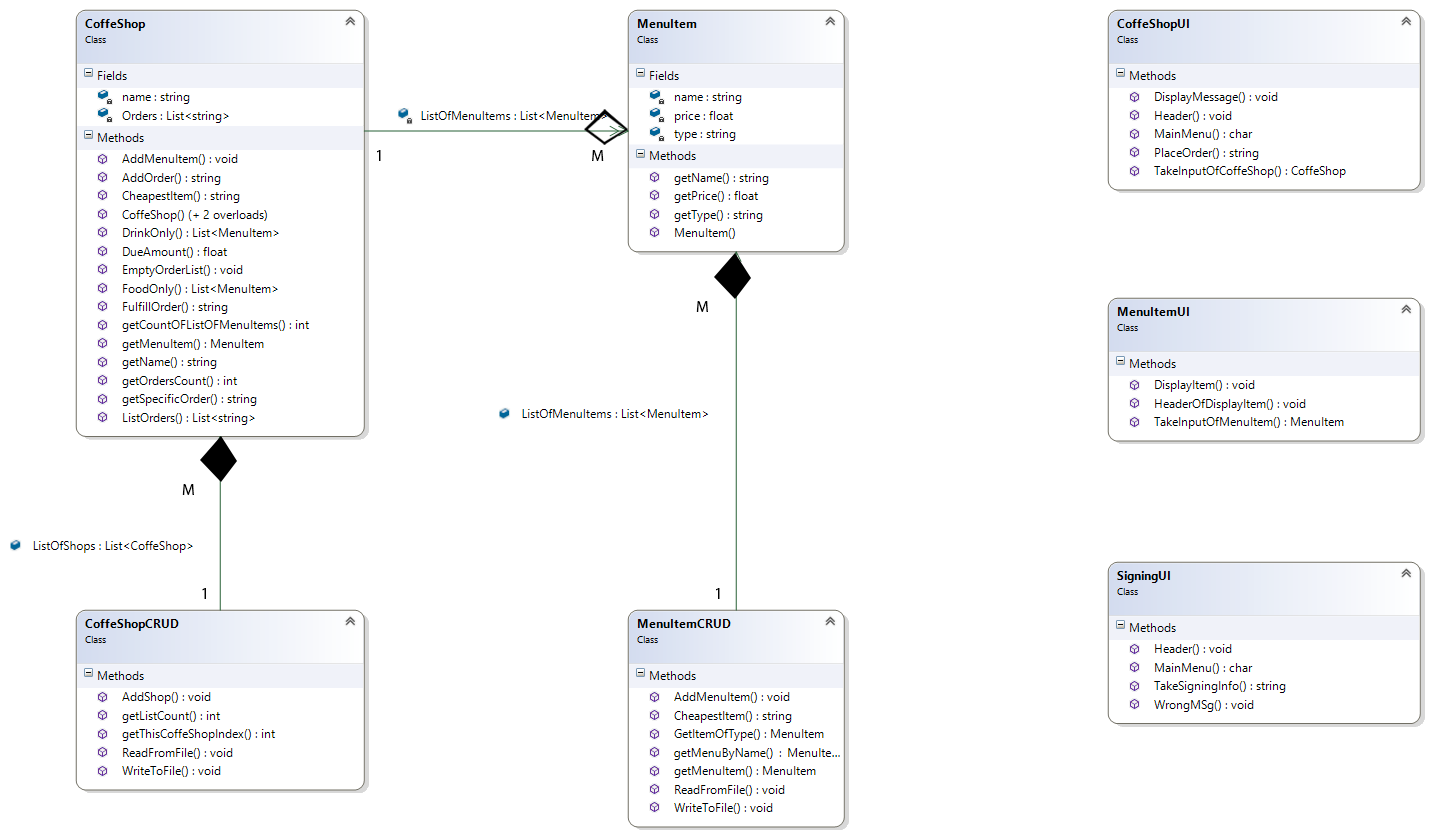
**Domain Model with Relations and Constraints**

****

**Domain Model with Multiplicity**



**Class Diagram**



**Code**

**BL:**

**CoffeeShop.cs**

class CoffeShop

{

public CoffeShop(string name)

{

this.name = name;

}

public CoffeShop(string name, List<MenuItem> ListOfMenuItems)

{

this.name = name;

this.ListOfMenuItems = ListOfMenuItems;

}

public CoffeShop(string name, List<MenuItem> ListOfMenuItems, List<string> Orders)

{

this.name = name;

this.ListOfMenuItems = ListOfMenuItems;

this.Orders = Orders;

}

private string name;

private List<MenuItem> ListOfMenuItems = new List<MenuItem>();

private List<string> Orders = new List<string>();

public void AddMenuItem(MenuItem Source)

{

ListOfMenuItems.Add(Source);

MenuItemCRUD.AddMenuItem(Source);

}

public string AddOrder(string name)

{

for (int i = 0; i < ListOfMenuItems.Count; i++)

{

if (name == ListOfMenuItems[i].getName())

{

Orders.Add(name);

return "Item Has been Added!";

}

}

return "This Item Is not available!";

}

public string FulfillOrder(int index)

{

if (Orders.Count != 0)

{

string ReturnValue = Orders[index];

return "The " + ReturnValue + " is ready!";

}

else

return "All orders Have Been Fulfilled!";

}

public List<string> ListOrders()

{

return Orders;

}

public float DueAmount()

{

float Total = 0;

for (int i = 0; i < Orders.Count; i++)

{

for (int j = 0; j < ListOfMenuItems.Count; j++)

{

if (Orders[i] == ListOfMenuItems[j].getName())

{

Total = Total + ListOfMenuItems[j].getPrice();

}

}

}

return Total;

}

public string CheapestItem()

{

return MenuItemCRUD.CheapestItem(ListOfMenuItems);

}

public List<MenuItem> FoodOnly()

{

List<MenuItem> Foods = new List<MenuItem>();

for (int i = 0; i < ListOfMenuItems.Count; i++)

{

MenuItem name = MenuItemCRUD.GetItemOfType(i, "Food");

if (name != null)

Foods.Add(name);

}

return Foods;

}

public List<MenuItem> DrinkOnly()

{

List<MenuItem> Drinks = new List<MenuItem>();

for (int i = 0; i < ListOfMenuItems.Count; i++)

{

MenuItem temp = MenuItemCRUD.GetItemOfType(i, "Drink");

if (temp != null)

Drinks.Add(temp);

}

return Drinks;

}

public string getName()

{

return name;

}

public int getCountOFListOFMenuItems()

{

return ListOfMenuItems.Count();

}

public MenuItem getMenuItem(int index)

{

return ListOfMenuItems[index];

}

public int getOrdersCount()

{

return Orders.Count;

}

public string getSpecificOrder(int index)

{

return Orders[index];

}

public void EmptyOrderList()

{

for (int i = 0; i < Orders.Count; i++)

Orders.Clear();

}

}

**MenuItem.cs**

class MenuItem

{

public MenuItem(string name, string type, float price)

{

this.name = name;

this.type = type;

this.price = price;

}

private string name;

private string type;

private float price;

public string getName()

{

return name;

}

public float getPrice()

{

return price;

}

public string getType()

{

return type;

}

}

**DL**

**CoffeeShopCRUD.cs**

class CoffeShopCRUD

{

public static List<CoffeShop> ListOfShops = new List<CoffeShop>();

public static void AddShop(CoffeShop source)

{

ListOfShops.Add(source);

}

public static int getListCount()

{

return ListOfShops.Count;

}

public static int getThisCoffeShopIndex(string name)

{

for (int i = 0; i < ListOfShops.Count; i++)

{

if (name == ListOfShops[i].getName())

return i;

}

return -1;

}

public static void WriteToFile()

{

string Path = "CoffeShop.txt";

StreamWriter File = new StreamWriter(Path);

for (int i = 0; i < ListOfShops.Count; i++)

{

File.Write(ListOfShops[i].getName() + ",");

if (ListOfShops[i].getCountOFListOFMenuItems() != 0)

{

for (int j = 0; j < ListOfShops[i].getCountOFListOFMenuItems() - 1; j++)

{

File.Write(ListOfShops[i].getMenuItem(j).getName() + ";");

}

File.Write(ListOfShops[i].getMenuItem(ListOfShops[i].getCountOFListOFMenuItems() - 1).getName() + ",");

}

else

{

File.Write("-1,");

}

if (ListOfShops[i].getOrdersCount() != 0)

{

for (int j = 0; j < ListOfShops[i].getOrdersCount() - 1; j++)

{

File.Write(ListOfShops[i].getSpecificOrder(j) + ";");

}

File.Write(ListOfShops[i].getSpecificOrder(ListOfShops[i].getOrdersCount() - 1));

}

else

{

File.Write("-1");

}

File.Flush();

File.WriteLine();

}

File.Close();

}

public static void ReadFromFile()

{

string Path = "CoffeShop.txt";

StreamReader File = new StreamReader(Path);

string Temp = "";

CoffeShop shop = null;

while ((Temp = File.ReadLine()) != null)

{

string[] SeparatedFields = Temp.Split(',');

string[] MenuItems = SeparatedFields[1].Split(';');

string[] order = SeparatedFields[2].Split(';');

Console.WriteLine(SeparatedFields[0] + " " + SeparatedFields[1] + " " + SeparatedFields[2]);

if (SeparatedFields[1] == "-1" && SeparatedFields[2] == "-1")

{

shop = new CoffeShop(SeparatedFields[0]);

}

else if (SeparatedFields[1] != "-1" && SeparatedFields[2] == "-1")

{

List<MenuItem> ListOfItems = new List<MenuItem>();

for (int j = 0; j < MenuItems.Length; j++)

{

MenuItem tempItem = MenuItemCRUD.getMenuByName(MenuItems[j]);

if (tempItem != null)

ListOfItems.Add(tempItem);

}

shop = new CoffeShop(SeparatedFields[0], ListOfItems);

}

else if (SeparatedFields[1] != "-1" && SeparatedFields[2] != "-1")

{

List<MenuItem> ListOfItems = new List<MenuItem>();

for (int j = 0; j < MenuItems.Length; j++)

{

MenuItem tempItem = MenuItemCRUD.getMenuByName(MenuItems[j]);

if (tempItem != null)

ListOfItems.Add(tempItem);

}

List<string> Orders = new List<string>();

for (int j = 0; j < order.Length; j++)

{

Orders.Add(order[j]);

}

shop = new CoffeShop(SeparatedFields[0], ListOfItems, Orders);

}

ListOfShops.Add(shop);

}

File.Close();

}

}

**MenuItemCRUD.cs**

class MenuItemCRUD

{

public static List<MenuItem> ListOfMenuItems = new List<MenuItem>();

public static void AddMenuItem(MenuItem Source)

{

ListOfMenuItems.Add(Source);

}

public static MenuItem getMenuItem(int index)

{

return ListOfMenuItems[index];

}

public static string CheapestItem(List<MenuItem> ListOfMenuItems)

{

float CheapestItemPrice;

int CheapestItemIndex;

CheapestItemIndex = 0;

CheapestItemPrice = ListOfMenuItems[0].getPrice();

for (int i = 1; i < ListOfMenuItems.Count; i++)

{

if (CheapestItemPrice > ListOfMenuItems[i].getPrice())

{

CheapestItemIndex = i;

CheapestItemPrice = ListOfMenuItems[i].getPrice();

}

}

return ListOfMenuItems[CheapestItemIndex].getName();

}

public static MenuItem GetItemOfType(int index, string type)

{

if (ListOfMenuItems[index].getType() == type)

return ListOfMenuItems[index];

return null;

}

public static void WriteToFile()

{

string Path = "MenuItem.txt";

StreamWriter File = new StreamWriter(Path);

for (int i = 0; i < ListOfMenuItems.Count; i++)

{

File.WriteLine(ListOfMenuItems[i].getName() + "," + ListOfMenuItems[i].getType() + "," + ListOfMenuItems[i].getPrice());

File.Flush();

}

File.Close();

}

public static void ReadFromFile()

{

string Path = "MenuItem.txt";

StreamReader File = new StreamReader(Path);

string Temp = "";

while ((Temp = File.ReadLine()) != null)

{

string[] SeparatedFields = Temp.Split(',');

MenuItem item = new MenuItem(SeparatedFields[0], SeparatedFields[1], float.Parse(SeparatedFields[2]));

ListOfMenuItems.Add(item);

}

File.Close();

}

public static MenuItem getMenuByName(string name)

{

for (int i = 0; i < ListOfMenuItems.Count; i++)

{

if (name == ListOfMenuItems[i].getName())

return ListOfMenuItems[i];

}

return null;

}

}

**UI**

**CoffeeShopUI.cs**

class CoffeShopUI

{

public static void Header()

{

Console.Clear();

Console.WriteLine("##############################################");

Console.WriteLine(" Welcome To Coffe Shop ");

Console.WriteLine("##############################################");

}

public static CoffeShop TakeInputOfCoffeShop()

{

CoffeShop temp;

Console.WriteLine("ENter Name of Coffe SHop....");

string name;

name = Console.ReadLine();

temp = new CoffeShop(name);

return temp;

}

public static char MainMenu()

{

Console.WriteLine("Press 1. To Add a Menu item");

Console.WriteLine("Press 2.To View the Cheapest Item in the menu");

Console.WriteLine("Press 3.To View the Drink’s Menu");

Console.WriteLine("Press 4.To View the Food’s Menu");

Console.WriteLine("Press 5.To Add Order");

Console.WriteLine("Press 6.To Fulfill the Order");

Console.WriteLine("Press 7.To View The orders List");

Console.WriteLine("Press 8.To Total Payable Amount");

Console.WriteLine("Press 9.To Exit");

char option;

char.TryParse(Console.ReadLine(), out option);

return option;

}

public static string PlaceOrder()

{

string NameOfItem;

Console.WriteLine();

Console.WriteLine("Enter Name Of item.....");

NameOfItem = Console.ReadLine();

return NameOfItem;

}

public static void DisplayMessage(string Msg)

{

Console.WriteLine(Msg);

}

}

**MenuItemUI.cs**

class MenuItemUI

{

public static MenuItem TakeInputOfMenuItem()

{

string name;

string type;

float Price;

Console.WriteLine("Enter NAme of Item...");

name = Console.ReadLine();

Console.WriteLine("ENter type of Item....");

type = Console.ReadLine();

Console.WriteLine("Enter Price Of Item...");

Price = float.Parse(Console.ReadLine());

MenuItem temp = new MenuItem(name, type, Price);

return temp;

}

public static void HeaderOfDisplayItem()

{

int Y = Console.CursorTop;

Console.Write("Name");

Console.SetCursorPosition(20, Y);

Console.Write("Category");

Console.SetCursorPosition(40, Y);

Console.WriteLine("Price");

}

public static void DisplayItem(MenuItem Source)

{

int Y = Console.CursorTop;

Console.Write(Source.getName());

Console.SetCursorPosition(20, Y);

Console.Write(Source.getType());

Console.SetCursorPosition(40, Y);

Console.WriteLine(Source.getPrice());

}}

**SigningUI.cs**

class SigningUI

{

public static void Header()

{

Console.Clear();

Console.WriteLine("##############################################");

Console.WriteLine(" Welcome To Coffe Shop ");

Console.WriteLine("##############################################");

}

public static char MainMenu()

{

Console.WriteLine("Press 1. To Sign in Your Shop");

Console.WriteLine("Press 2. To Sign up a Shop");

char option;

char.TryParse(Console.ReadLine(), out option);

return option;

}

public static string TakeSigningInfo()

{

Console.WriteLine("Enter NAme of Your Shop.....");

return Console.ReadLine();

}

public static void WrongMSg()

{

Console.WriteLine("Wrong NAme.....");

Console.ReadLine();

}

}

**Driver Program**

class Program

{

static void SigningOfUser(ref int ShopIndex)

{

char SignOption;

SignOption = SigningUI.MainMenu();

if (SignOption == '2')

{

CoffeShopCRUD.AddShop(CoffeShopUI.TakeInputOfCoffeShop());

ShopIndex = (CoffeShopCRUD.getListCount() - 1);

CoffeShopCRUD.WriteToFile();

}

else if (SignOption == '1')

{

ShopIndex = CoffeShopCRUD.getThisCoffeShopIndex(SigningUI.TakeSigningInfo());

if (ShopIndex == -1)

{

SigningUI.WrongMSg();

}

}

}

static void Option\_1(int ShopIndex)

{

CoffeShopUI.Header();

MenuItem Input;

Input = MenuItemUI.TakeInputOfMenuItem();

CoffeShopCRUD.ListOfShops[ShopIndex].AddMenuItem(Input);

MenuItemCRUD.WriteToFile();

CoffeShopCRUD.WriteToFile();

}

static void Option\_2(int ShopIndex)

{

CoffeShopUI.Header();

CoffeShopUI.DisplayMessage("The CHeapest Item is " + CoffeShopCRUD.ListOfShops[ShopIndex].CheapestItem());

}

static void Option\_3(int ShopIndex)

{

CoffeShopUI.Header();

MenuItemUI.HeaderOfDisplayItem();

List<MenuItem> ListOfDrinks = CoffeShopCRUD.ListOfShops[ShopIndex].DrinkOnly();

for (int i = 0; i < ListOfDrinks.Count; i++)

{

MenuItemUI.DisplayItem(ListOfDrinks[i]);

}

}

static void Option\_4(int ShopIndex)

{

CoffeShopUI.Header();

MenuItemUI.HeaderOfDisplayItem();

List<MenuItem> ListOfFood = CoffeShopCRUD.ListOfShops[ShopIndex].FoodOnly();

for (int i = 0; i < ListOfFood.Count; i++)

{

MenuItemUI.DisplayItem(ListOfFood[i]);

}

}

static void Option\_5(int ShopIndex)

{

CoffeShopUI.Header();

MenuItemUI.HeaderOfDisplayItem();

for (int i = 0; i < CoffeShopCRUD.ListOfShops[ShopIndex].getCountOFListOFMenuItems(); i++)

{

MenuItemUI.DisplayItem(CoffeShopCRUD.ListOfShops[ShopIndex].getMenuItem(i));

}

string Msg = CoffeShopCRUD.ListOfShops[ShopIndex].AddOrder(CoffeShopUI.PlaceOrder());

CoffeShopUI.DisplayMessage(Msg);

MenuItemCRUD.WriteToFile();

CoffeShopCRUD.WriteToFile();

}

static void Option\_6(int ShopIndex)

{

CoffeShopUI.Header();

int y;

for (int i = 0; i < CoffeShopCRUD.ListOfShops[ShopIndex].getOrdersCount(); i++)

{

string msg;

msg = CoffeShopCRUD.ListOfShops[ShopIndex].FulfillOrder(i);

CoffeShopUI.DisplayMessage(msg);

}

CoffeShopCRUD.ListOfShops[ShopIndex].EmptyOrderList();

CoffeShopCRUD.WriteToFile();

}

static void Option\_7(int ShopIndex)

{

CoffeShopUI.Header();

List<string> Orders = CoffeShopCRUD.ListOfShops[ShopIndex].ListOrders();

CoffeShopUI.DisplayMessage("Name");

for (int i = 0; i < Orders.Count; i++)

{

CoffeShopUI.DisplayMessage((i + 1).ToString() + " " + Orders[i]);

}

}

static void Option\_8(int ShopIndex)

{

CoffeShopUI.Header();

CoffeShopUI.DisplayMessage("The Total Payable Amount is = " + CoffeShopCRUD.ListOfShops[ShopIndex].DueAmount().ToString());

}

static void Main(string[] args)

{

MenuItemCRUD.ReadFromFile();

CoffeShopCRUD.ReadFromFile();

again:;

int ShopIndex = 0;

SigningUI.Header();

SigningOfUser(ref ShopIndex);

if (ShopIndex == -1)

goto again;

CoffeShopUI.Header();

char option;

do

{

CoffeShopUI.Header();

option = CoffeShopUI.MainMenu();

switch (option)

{

case '1':

Option\_1(ShopIndex);

break;

case '2':

Option\_2(ShopIndex);

break;

case '3':

Option\_3(ShopIndex);

break;

case '4':

Option\_4(ShopIndex);

break;

case '5':

Option\_5(ShopIndex);

break;

case '6':

Option\_6(ShopIndex);

break;

case '7':

Option\_7(ShopIndex);

break;

case '8':

Option\_8(ShopIndex);

break;

case '9':

CoffeShopUI.DisplayMessage("Thanks For Using Application");

break;

default:

CoffeShopUI.DisplayMessage("Wrong Option");

break;

}

Console.ReadKey();

} while (option != '9');

}

}

**Sequence Diagrams**

**Game Object**

**Case Study**

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).

● 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

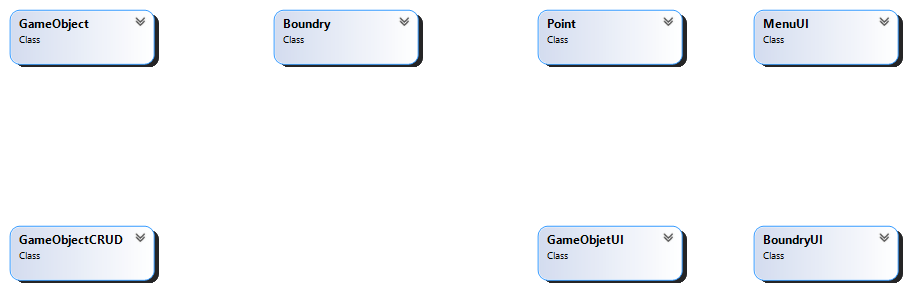
direction. For example, if the direction is from left to right it will move the game object

one step toward right.

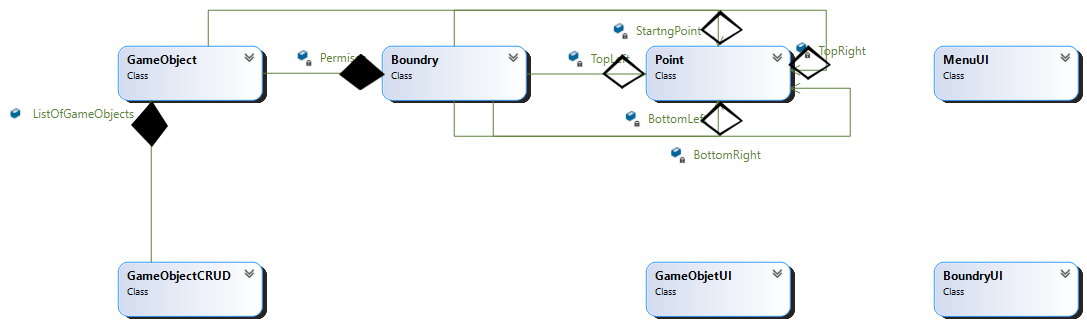
○ **Erase**: When called, this method will erase the shape on the console.

○ **Draw:** When called, this method will draw the shape on the console.

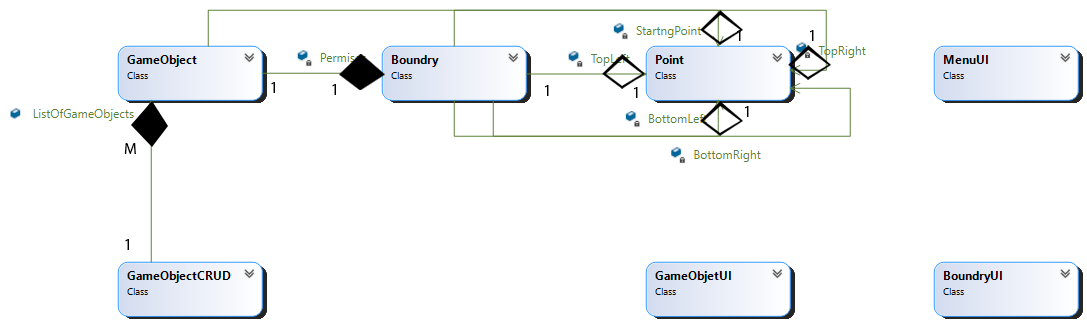
**Domain Model With Class Names**



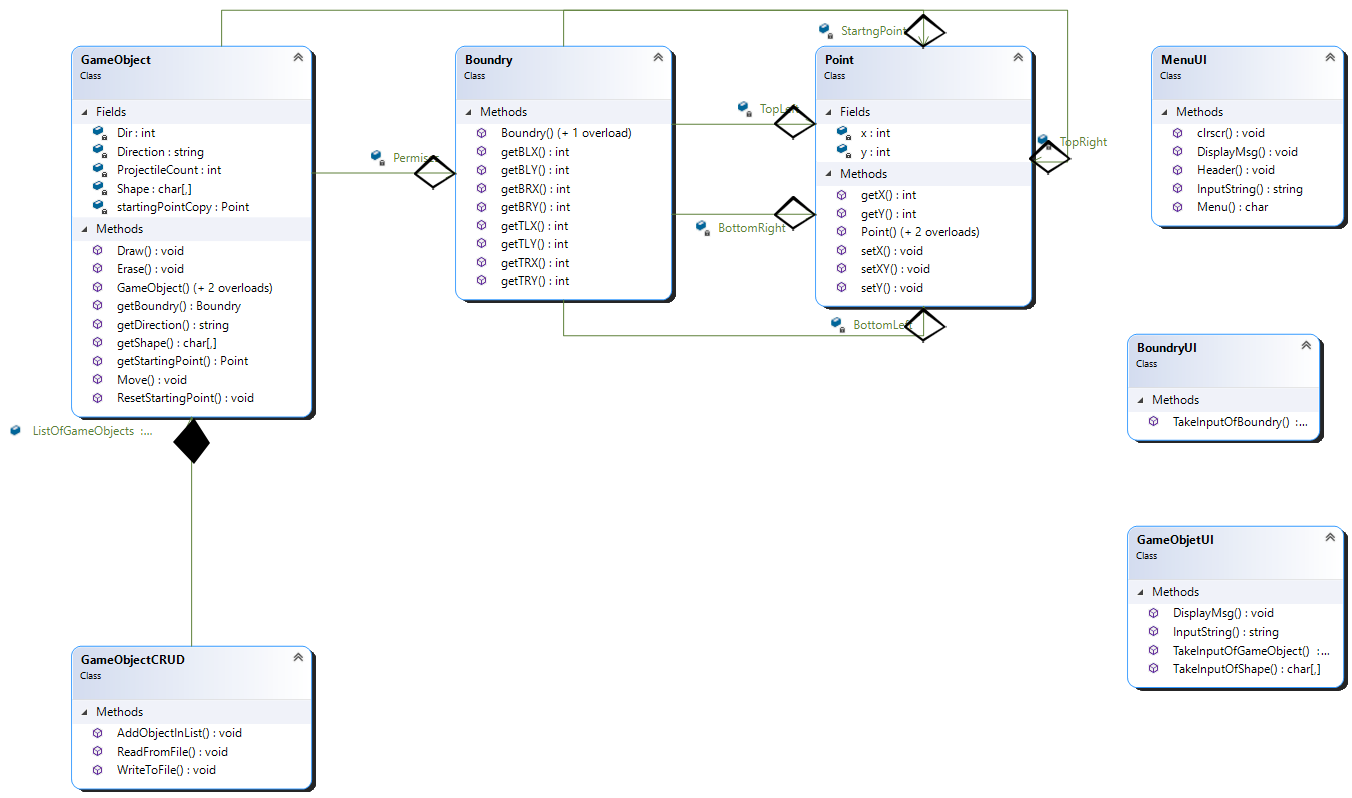
**Domain Model With Relations And Constraints**



**Domain Model With Multiplicity**



**Class Diagram**



**Complete Code**

**BL:**

**Boundry:**

class Boundry

{

public Boundry()

{

TopLeft = new Point(0, 0);

TopRight = new Point(90, 0);

BottomLeft = new Point(0, 90);

BottomRight = new Point(90, 90);

}

public Boundry(Point TopLeft, Point TopRight, Point BottomLeft, Point BottomRight)

{

this.TopLeft = TopLeft;

this.TopRight = TopRight;

this.BottomLeft = BottomLeft;

this.BottomRight = BottomRight;

}

private Point TopLeft;

private Point TopRight;

private Point BottomLeft;

private Point BottomRight;

public int getTLX()

{

return TopLeft.getX();

}

public int getTLY()

{

return TopLeft.getY();

}

public int getBLX()

{

return BottomLeft.getX();

}

public int getBLY()

{

return BottomLeft.getY();

}

public int getTRX()

{

return TopRight.getX();

}

public int getTRY()

{

return TopRight.getY();

}

public int getBRX()

{

return BottomRight.getX();

}

public int getBRY()

{

return BottomRight.getY();

}

}

**GameObjects:**

class GameObject

{

public GameObject()

{

Shape = new char[1, 3] { { '-', '-', '-' } };

StartngPoint = new Point();

Permises = new Boundry();

Direction = "LeftToRight";

}

public GameObject(char[,] Shape, Point StartingPoint)

{

this.Shape = Shape;

this.StartngPoint = StartingPoint;

Permises = new Boundry();

Direction = "Diagonal";

}

public GameObject(char[,] Shape, Point StartingPoint, Boundry Permises, string Direction)

{

this.Shape = Shape;

this.StartngPoint = StartingPoint;

this.Permises = Permises;

this.Direction = Direction;

startingPointCopy = new Point(StartngPoint);

}

private char[,] Shape;

private Point StartngPoint;

private Point startingPointCopy;

private Boundry Permises;

private string Direction;

private int Dir = 1;//Patrol controller

private int ProjectileCount = 0;

public void Draw(bool erase = false)

{

bool flagOfShapePrint = false;

for (int i = Permises.getTLY(); i <= Permises.getBLY(); i++)

{

for (int j = Permises.getTLX(); j <= Permises.getTRX(); j++)

{

if (j == StartngPoint.getX() && i == StartngPoint.getY())

flagOfShapePrint = true;

if (flagOfShapePrint)

{

Console.SetCursorPosition(j, i);

for (int k = 0; k < Shape.GetLength(0); k++)

{

for (int l = 0; l < Shape.GetLength(1); l++)

{

if (erase)

{

GameObjetUI.DisplayMsg(" ");

}

else

{

GameObjetUI.DisplayMsg(Shape[k, l].ToString());

}

}

GameObjetUI.DisplayMsg("\n");

i++;

Console.SetCursorPosition(j, i);

}

flagOfShapePrint = false;

break;

}

}

}

}

public void Move()

{

switch (Direction)

{

case "LeftToRight":

if (StartngPoint.getX() != Permises.getTRX())

StartngPoint.setX(StartngPoint.getX() + 1);

break;

case "RightToLeft":

if (StartngPoint.getX() != Permises.getTLX())

StartngPoint.setX(StartngPoint.getX() - 1);

break;

case "Patrol":

if (Dir == 1)

StartngPoint.setX(StartngPoint.getX() - 1);

if (Dir == 2)

StartngPoint.setX(StartngPoint.getX() + 1);

if (StartngPoint.getX() == Permises.getBLX())

Dir = 2;

if (StartngPoint.getX() == Permises.getBRX())

Dir = 1;

break;

case "Projectile":

{

if (ProjectileCount >= 0 && ProjectileCount <= 5)

{

StartngPoint.setX(StartngPoint.getX() + 1);

StartngPoint.setY(StartngPoint.getY() - 1);

}

if (ProjectileCount == 6 || ProjectileCount == 7)

{

StartngPoint.setX(StartngPoint.getX() + 1);

}

if (ProjectileCount >= 8 && ProjectileCount <= 11)

{

StartngPoint.setX(StartngPoint.getX() + 1);

StartngPoint.setY(StartngPoint.getY() + 1);

}

ProjectileCount++;

if (ProjectileCount == 11)

ProjectileCount = 0;

}

break;

case "Diagonal":

if (StartngPoint.getX() != Permises.getTRX())

{

StartngPoint.setX(StartngPoint.getX() + 1);

StartngPoint.setY(StartngPoint.getY() + 1);

}

break;

}

}

public void Erase()

{

Draw(true);

}

public void ResetStartingPoint()

{

StartngPoint.setX(startingPointCopy.getX());

StartngPoint.setY(startingPointCopy.getY());

}

public char[,] getShape()

{

return Shape;

}

public Point getStartingPoint()

{

return StartngPoint;

}

public Boundry getBoundry()

{

return Permises;

}

public string getDirection()

{

return Direction;

}

}

**Point:**

class Point

{

public Point(Point p)

{

this.x = p.getX();

this.y = p.getY();

}

public Point()

{

}

public Point(int x, int y)

{

this.x = x;

this.y = y;

}

private int x;

private int y;

public void setX(int x)

{

this.x = x;

}

public void setY(int y)

{

this.y = y;

}

public int getX()

{

return x;

}

public int getY()

{

return y;

}

public void setXY(int x, int y)

{

this.x = x;

this.y = y;

}

}

**DL:**

**GameObjectCRUD:**

class GameObjectCRUD

{

public static List<GameObject> ListOfGameObjects = new List<GameObject>();

public static void AddObjectInList(GameObject Source)

{

ListOfGameObjects.Add(Source);

}

public static void WriteToFile()

{

string path = "Objects.txt";

StreamWriter File = new StreamWriter(path);

for (int i = 0; i < ListOfGameObjects.Count; i++)

{

char[,] shape = ListOfGameObjects[i].getShape();

File.Write(shape.GetLength(0) + "," + shape.GetLength(1) + ",");

for (int j = 0; j < shape.GetLength(0) - 1; j++)

{

for (int k = 0; k < shape.GetLength(1); k++)

{

File.Write(shape[j, k].ToString() + ";");

}

}

for (int j = 0; j < shape.GetLength(1) - 1; j++)

{

File.Write(shape[shape.GetLength(0) - 1, j].ToString() + ";");

}

File.Write(shape[shape.GetLength(0) - 1, shape.GetLength(1) - 1].ToString() + ",");

Point StartingPoint = ListOfGameObjects[i].getStartingPoint();

File.Write(StartingPoint.getX().ToString() + ";" + StartingPoint.getY().ToString() + ",");

Boundry permises = ListOfGameObjects[i].getBoundry();

File.Write(permises.getTLX() + ";" + permises.getTLY() + ";" + permises.getTRX() + ";" + permises.getTRY() + ";" + permises.getBLX() + ";" + permises.getBLY() + ";" + permises.getBRX() + ";" + permises.getBRY() + ",");

File.WriteLine(ListOfGameObjects[i].getDirection());

File.Flush();

}

File.Close();

}

public static void ReadFromFile()

{

string path = "Objects.txt";

StreamReader File = new StreamReader(path);

string temp = "";

while ((temp = File.ReadLine()) != null)

{

char[,] Shape;

Point StartingPoint;

Boundry Permises;

string[] fields = temp.Split(',');

Shape = new char[int.Parse(fields[0]), int.Parse(fields[1])];

string[] shapeFields = fields[2].Split(';');

int count = 0;

for (int i = 0; i < int.Parse(fields[0]); i++)

{

for (int j = 0; j < int.Parse(fields[1]); j++)

{

Shape[i, j] = char.Parse(shapeFields[count]);

count++;

}

}

string[] SP\_Fields = fields[3].Split(';');

StartingPoint = new Point(int.Parse(SP\_Fields[0]), int.Parse(SP\_Fields[1]));

string[] Permises\_Fields = fields[4].Split(';');

Permises = new Boundry(new Point(int.Parse(Permises\_Fields[0]), int.Parse(Permises\_Fields[1])), new Point(int.Parse(Permises\_Fields[2]), int.Parse(Permises\_Fields[3])), new Point(int.Parse(Permises\_Fields[4]), int.Parse(Permises\_Fields[5])), new Point(int.Parse(Permises\_Fields[6]), int.Parse(Permises\_Fields[7])));

GameObject Source = new GameObject(Shape, StartingPoint, Permises, fields[5]);

ListOfGameObjects.Add(Source);

}

File.Close();

}

}

**UI:**

**BoundryUI:**

class BoundryUI

{

public static Boundry TakeInputOfBoundry()

{

int x, y;

Point TL;

Point TR;

Point BL;

Point BR;

Console.WriteLine("ENter X-cordinate of Top Left Point = ");

x = int.Parse(Console.ReadLine());

Console.WriteLine("ENter Y-cordinate of Top Left Point = ");

y = int.Parse(Console.ReadLine());

TL = new Point(x, y);

Console.WriteLine("ENter X-cordinate of Top Right Point = ");

x = int.Parse(Console.ReadLine());

Console.WriteLine("ENter Y-cordinate of Top Right Point = ");

y = int.Parse(Console.ReadLine());

TR = new Point(x, y);

Console.WriteLine("ENter X-cordinate of Bottom Left Point = ");

x = int.Parse(Console.ReadLine());

Console.WriteLine("ENter Y-cordinate of Bottom Left Point = ");

y = int.Parse(Console.ReadLine());

BL = new Point(x, y);

Console.WriteLine("ENter X-cordinate of Bottom Right Point = ");

x = int.Parse(Console.ReadLine());

Console.WriteLine("ENter Y-cordinate of Bottom Right Point = ");

y = int.Parse(Console.ReadLine());

BR = new Point(x, y);

Boundry temp = new Boundry(TL, TR, BL, BR);

return temp;

}

}

**GameObjectUI:**class GameObjetUI

{

public static char[,] TakeInputOfShape()

{

int row, col;

Console.WriteLine("NEter no of Rows = ");

row = int.Parse(Console.ReadLine());

Console.WriteLine("Enter No of Columns = ");

col = int.Parse(Console.ReadLine());

char[,] input = new char[row, col];

for (int i = 0; i < row; i++)

{

for (int j = 0; j < col; j++)

{

Console.Write("Enter {0} row and {1} column item = ", i + 1, j + 1);

char.TryParse(Console.ReadLine(), out input[i, j]);

}

}

return input;

}

public static GameObject TakeInputOfGameObject()

{

char[,] shape = TakeInputOfShape();

Boundry permises = BoundryUI.TakeInputOfBoundry();

int x, y;

Console.WriteLine("ENter X-cordinate of Starting Point = ");

x = int.Parse(Console.ReadLine());

Console.WriteLine("ENter Y-cordinate of Starting Point = ");

y = int.Parse(Console.ReadLine());

Point StartingPoint = new Point(x, y);

Console.WriteLine("Enter Direction");

string Direction = Console.ReadLine();

GameObject temp = new GameObject(shape, StartingPoint, permises, Direction);

return temp;

}

public static void DisplayMsg(string Msg)

{

Console.Write(Msg);

}

public static string InputString()

{

return Console.ReadLine();

}

}

**MenuUI:**

class MenuUI

{

public static void clrscr()

{

Console.Clear();

}

public static void Header()

{

Console.Clear();

Console.WriteLine("##################################");

Console.WriteLine("# ...Game Object... #");

Console.WriteLine("##################################");

}

public static void DisplayMsg(string Msg)

{

Console.Write(Msg);

}

public static string InputString()

{

return Console.ReadLine();

}

public static char Menu()

{

Console.WriteLine("Press 1. To Add Game Object");

Console.WriteLine("Press 2. To View All Game Objects");

Console.WriteLine("Press 3. To View Moving Game Objects");

Console.WriteLine("Press 4. To Erase All Objects");

Console.WriteLine("Press 5. To Exit");

char option;

char.TryParse(Console.ReadLine(), out option);

return option;

}

}

**Driver Program**

class Program

{

static void Option\_1()

{

MenuUI.Header();

GameObjectCRUD.AddObjectInList(GameObjetUI.TakeInputOfGameObject());

GameObjectCRUD.WriteToFile();

}

static void Option\_2()

{

MenuUI.clrscr();

for (int i = 0; i < GameObjectCRUD.ListOfGameObjects.Count; i++)

{

GameObjectCRUD.ListOfGameObjects[i].Draw();

}

MenuUI.InputString();

}

static void Option\_3()

{

MenuUI.clrscr();

bool flag = true;

while (flag)

{

for (int i = 0; i < GameObjectCRUD.ListOfGameObjects.Count; i++)

{

GameObjectCRUD.ListOfGameObjects[i].Draw();

}

Thread.Sleep(100);

for (int i = 0; i < GameObjectCRUD.ListOfGameObjects.Count; i++)

{

GameObjectCRUD.ListOfGameObjects[i].Erase();

}

for (int i = 0; i < GameObjectCRUD.ListOfGameObjects.Count; i++)

{

GameObjectCRUD.ListOfGameObjects[i].Move();

}

if (EZInput.Keyboard.IsKeyPressed(Key.Escape))

flag = false;

}

for (int i = 0; i < GameObjectCRUD.ListOfGameObjects.Count; i++)

{

GameObjectCRUD.ListOfGameObjects[i].ResetStartingPoint();

}

MenuUI.InputString();

}

static void Option\_4()

{

for (int i = 0; i < GameObjectCRUD.ListOfGameObjects.Count; i++)

{

GameObjectCRUD.ListOfGameObjects[i].Erase();

}

}

static void Main(string[] args)

{

GameObjectCRUD.ReadFromFile();

MenuUI.Header();

char option;

do

{

MenuUI.Header();

option = MenuUI.Menu();

if (option == '1')

{

Option\_1();

}

else if (option == '2')

{

Option\_2();

}

else if (option == '3')

{

Option\_3();

}

else if (option == '4')

{

Option\_4();

}

MenuUI.InputString();

} while (option != '5');

}

}

**Sequence Diagram**

**Ocean Navigation**

**Case Study**

In ocean navigation, locations are measured in degrees and minutes of latitude and longitude. Thus if

you’re lying off the mouth of Papeete Harbor in Tahiti, your location is 149 degrees 34.8 minutes west

longitude, and 17 degrees 31.5 minutes south latitude. This is written as 149°34.8’ W, 17°31.5’ S. There

are 60 minutes in a degree. (An older system also divided a minute into 60 seconds, but the modern

approach is to use decimal minutes instead.) Longitude is measured from 0 to 180 degrees, east or west

from Greenwich, England, to the international dateline in the Pacific. Latitude is measured from 0 to 90

degrees, north or south from the equator to the poles.

Create a class angle that includes three member variables: an int for degrees, a float for minutes, and a

char for the direction letter (N, S, E, or W). This class can hold either a latitude variable or a longitude

variable.

Write one member function to change the angle value (in degrees and minutes) and a direction given

from the user, and a second to display the angle value in 179°59.9’ E in string format. Also write a

three-argument constructor.

You can use this to print a degree (°) symbol.

Console.WriteLine("\u00b0");

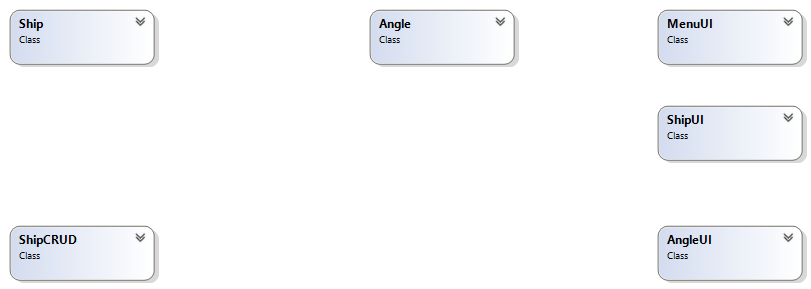
Create a class called ship that incorporates a ship’s number and location. Use two variables of the angle

class to represent the ship’s latitude and longitude for the ship’s location. Write a parameterized

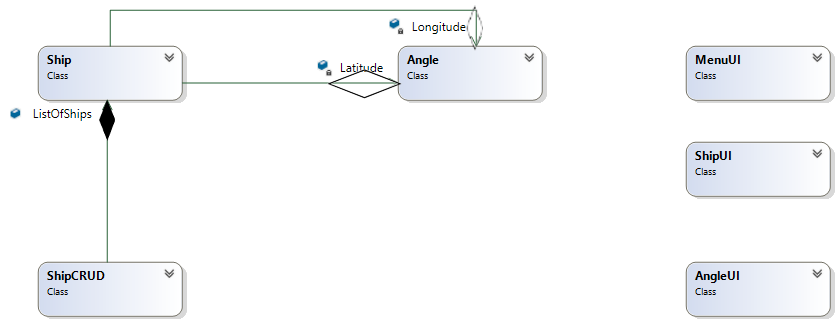
constructor to initialize the attributes of the ship class. A member function of the ship class should print

the position (latitude and longitude) of the ship; another should report/print the serial number.

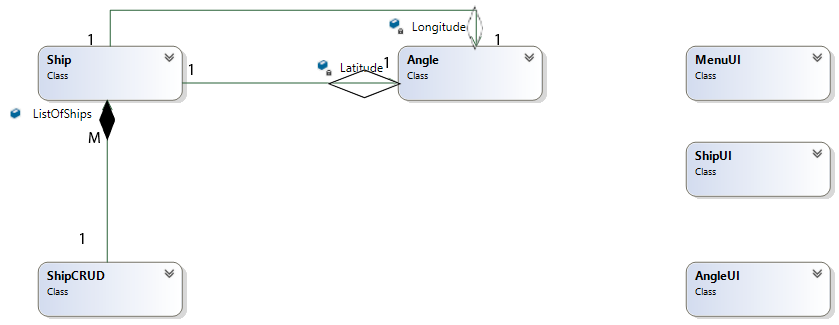
**Domain Model With Class Names**



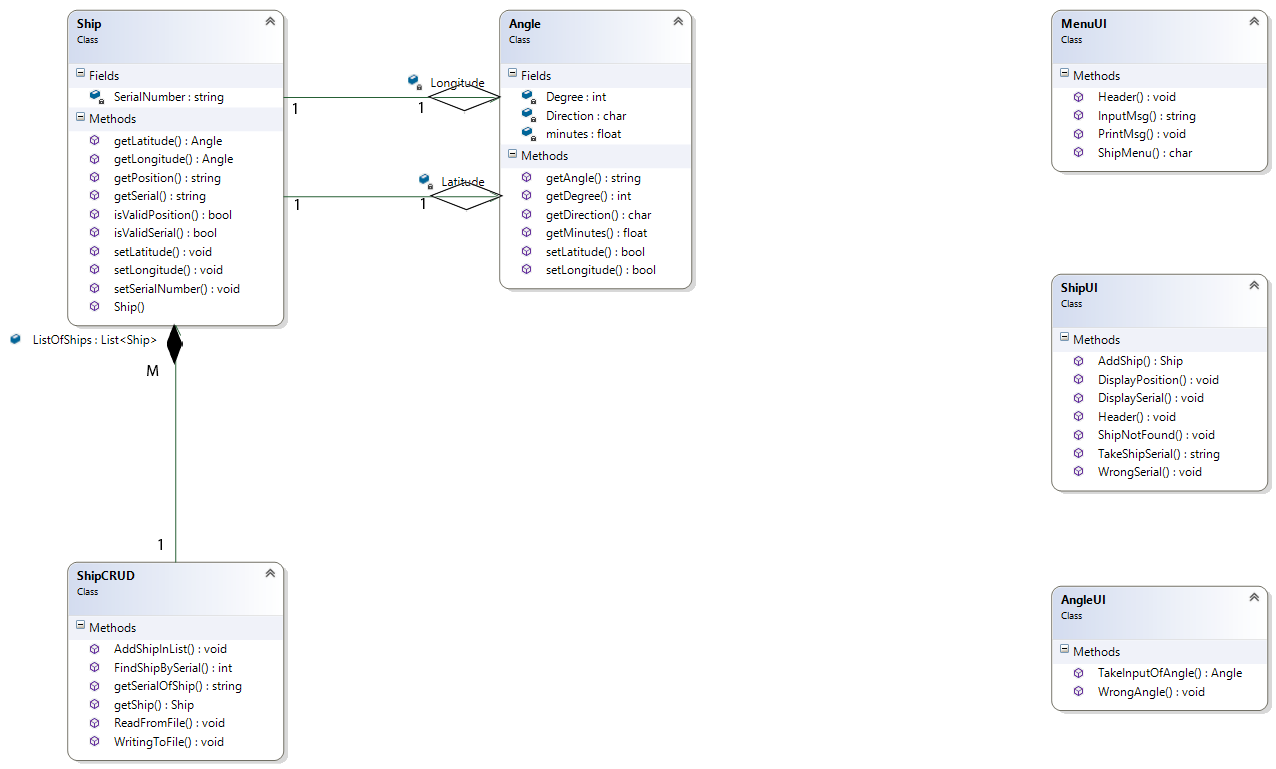
**Domain Model With Relations And Constraints**



**Domain Model With Multiplicity**



**Class Diagram**



**Complete Code**

**BL:**

**Angle:**

class Angle

{

private int Degree;

private float minutes;

private char Direction;

public bool setLongitude(int Degree, float minutes, char Direction)

{

if (Degree >= 0 && Degree <= 180)

if (Direction == 'E' || Direction == 'W')

{

this.Degree = Degree;

this.minutes = minutes;

this.Direction = Direction;

return true;

}

return false;

}

public bool setLatitude(int Degree, float minutes, char Direction)

{

if (Degree >= 0 && Degree <= 90)

if (Direction == 'N' || Direction == 'S')

{

this.Degree = Degree;

this.minutes = minutes;

this.Direction = Direction;

return true;

}

return false;

}

public string getAngle()

{

string output = Degree.ToString() + "\u00b0" + minutes.ToString() + "\'" + " " + Direction.ToString();

return output;

}

public int getDegree()

{

return Degree;

}

public float getMinutes()

{

return minutes;

}

public char getDirection()

{

return Direction;

}

}

**Ship:**

class Ship

{

public Ship(string SerialNumber, Angle Longitude, Angle Latitude)

{

this.Longitude = Longitude;

this.Latitude = Latitude;

this.SerialNumber = SerialNumber;

}

private Angle Longitude;

private Angle Latitude;

private string SerialNumber;

public void setLongitude(Angle Longitude)

{

this.Longitude = Longitude;

}

public void setLatitude(Angle Latitude)

{

this.Latitude = Latitude;

}

public void setSerialNumber(string SerialNumber)

{

this.SerialNumber = SerialNumber;

}

public string getPosition()

{

return "Ship is at " + Longitude.getAngle() + " and " + Latitude.getAngle();

}

public string getSerial()

{

return SerialNumber;

}

public bool isValidSerial(string serial)

{

if (serial == SerialNumber)

return true;

return false;

}

public bool isValidPosition(Angle Longitude, Angle Latitude)

{

if (Longitude.getDegree() == this.Longitude.getDegree() && Longitude.getMinutes() == this.Longitude.getMinutes() && Longitude.getDirection() == this.Longitude.getDirection() && Latitude.getDegree() == this.Latitude.getDegree() && Latitude.getMinutes() == this.Latitude.getMinutes() && Latitude.getDirection() == this.Latitude.getDirection())

return true;

return false;

}

public Angle getLongitude()

{

return Longitude;

}

public Angle getLatitude()

{

return Latitude;

}

}

**DL:**

**ShipCRUD:**

class ShipCRUD

{

public static List<Ship> ListOfShips = new List<Ship>();

public static void AddShipInList(Ship Source)

{

ListOfShips.Add(Source);

}

public static Ship getShip(int index)

{

return ListOfShips[index];

}

public static int FindShipBySerial(string SerialNumber)

{

for (int i = 0; i < ListOfShips.Count; i++)

{

if (SerialNumber == ListOfShips[i].getSerial())

return i;

}

return -1;

}

public static string getSerialOfShip(Angle Longitude, Angle Latitude)

{

for (int i = 0; i < ListOfShips.Count; i++)

{

if (ListOfShips[i].isValidPosition(Longitude, Latitude))

return ListOfShips[i].getSerial();

}

return null;

}

public static void ReadFromFile()

{

string path = "Ship.txt";

StreamReader File = new StreamReader(path);

string temp = " ";

while ((temp = File.ReadLine()) != null)

{

string[] fields = temp.Split(',');

string[] LongitudeFields = fields[1].Split(';');

string[] LatitudeFields = fields[2].Split(';');

Angle Longitude = new Angle();

Angle Latitude = new Angle();

Longitude.setLongitude(int.Parse(LongitudeFields[0]), float.Parse(LongitudeFields[1]), char.Parse(LongitudeFields[2]));

Latitude.setLatitude(int.Parse(LatitudeFields[0]), float.Parse(LatitudeFields[1]), char.Parse(LatitudeFields[2]));

AddShipInList(new Ship(fields[0], Longitude, Latitude));

}

File.Close();

}

public static void WritingToFile()

{

string path = "Ship.txt";

StreamWriter File = new StreamWriter(path);

for (int i = 0; i < ListOfShips.Count; i++)

{

Angle Longitude = ListOfShips[i].getLongitude();

Angle Latitude = ListOfShips[i].getLatitude();

File.WriteLine(ListOfShips[i].getSerial() + "," + Longitude.getDegree() + ";" + Longitude.getMinutes() + ";" + Longitude.getDirection() + "," + Latitude.getDegree() + ";" + Latitude.getMinutes() + ";" + Latitude.getDirection());

File.Flush();

}

File.Close();

}

}

**UI:**

**AngleUI:**

class AngleUI

{

public static Angle TakeInputOfAngle(string type)

{

Angle Temp = new Angle();

int Degree;

float Minutes;

char Direction;

Console.WriteLine("....." + type + ".....");

Console.WriteLine("ENter " + type + "'s Degree...");

Degree = int.Parse(Console.ReadLine());

Console.WriteLine("ENter " + type + "'s Minutes...");

Minutes = float.Parse(Console.ReadLine());

Console.WriteLine("ENter " + type + "'s Direction...");

char.TryParse(Console.ReadLine(), out Direction);

if (type == "Longitude")

{

if (Temp.setLongitude(Degree, Minutes, Direction))

return Temp;

}

if (type == "Latitude")

{

if (Temp.setLatitude(Degree, Minutes, Direction))

return Temp;

}

return null;

}

public static void WrongAngle()

{

Console.WriteLine("Wrong ANgle....");

}

}

**MenuUI:**

class MenuUI

{

public static void Header()

{

Console.Clear();

Console.WriteLine("||||||||||||||||||||||||||||||||||||||||||||||");

Console.WriteLine("| Ship Navigation System |");

Console.WriteLine("||||||||||||||||||||||||||||||||||||||||||||||");

}

public static char ShipMenu()

{

Console.WriteLine("1. Add Ship");

Console.WriteLine("2. View Ship Position");

Console.WriteLine("3. View Ship Serial Number");

Console.WriteLine("4. Change Ship Position");

Console.WriteLine("5. Exit");

char option;

char.TryParse(Console.ReadLine(), out option);

return option;

}

public static string InputMsg()

{

return Console.ReadLine();

}

public static void PrintMsg(string Msg)

{

Console.WriteLine(Msg);

}

}

**ShipUI:**

class ShipUI

{

public static string TakeShipSerial()

{

Console.WriteLine("Enter Serial Number Of Ship....");

string Serial = Console.ReadLine();

return Serial;

}

public static void Header()

{

Console.Clear();

Console.WriteLine("||||||||||||||||||||||||||||||||||||||||||||||");

Console.WriteLine("| Ship Navigation System |");

Console.WriteLine("||||||||||||||||||||||||||||||||||||||||||||||");

}

public static Ship AddShip()

{

string SerialNumber = "";

Angle Longitude;

Angle Latitude;

Ship Temp;

Console.WriteLine("ENter Ship Serial Number...");

SerialNumber = Console.ReadLine();

Label\_1:

Longitude = AngleUI.TakeInputOfAngle("Longitude");

if (Longitude == null)

{

Console.WriteLine("Wrong input...");

goto Label\_1;

}

Label\_2:

Latitude = AngleUI.TakeInputOfAngle("Latitude");

if (Latitude == null)

{

Console.WriteLine("Wrong input...");

goto Label\_2;

}

Temp = new Ship(SerialNumber, Longitude, Latitude);

return Temp;

}

public static void DisplayPosition(Ship Souce)

{

Console.WriteLine(Souce.getPosition());

}

public static void WrongSerial()

{

Console.WriteLine("wrong Serial");

}

public static void ShipNotFound()

{

Console.WriteLine("Ship Not Found");

}

public static void DisplaySerial(string SerialNumber)

{

Console.WriteLine("Ship Serial is " + SerialNumber);

}

}

**Driver Program**

class Program

{

public static int vari = 9;

static void option\_1()

{

ShipUI.Header();

ShipCRUD.ListOfShips.Add(ShipUI.AddShip());

ShipCRUD.WritingToFile();

}

static void option\_2()

{

ShipUI.Header();

int indexOfShip = ShipCRUD.FindShipBySerial(ShipUI.TakeShipSerial());

if (indexOfShip != -1)

ShipUI.DisplayPosition(ShipCRUD.ListOfShips[indexOfShip]);

else

ShipUI.WrongSerial();

}

static void option\_3()

{

ShipUI.Header();

Angle Longitude = AngleUI.TakeInputOfAngle("Longitude");

if (Longitude == null)

{

AngleUI.WrongAngle();

goto label;

}

Angle Latitude = AngleUI.TakeInputOfAngle("Latitude");

if (Longitude == null)

{

AngleUI.WrongAngle();

}

string SerialNumber = ShipCRUD.getSerialOfShip(Longitude, Latitude);

if (SerialNumber == null)

ShipUI.ShipNotFound();

else

ShipUI.DisplaySerial(SerialNumber);

label:;

}

static void option\_4()

{

ShipUI.Header();

string serial = ShipUI.TakeShipSerial();

int index = ShipCRUD.FindShipBySerial(serial);

if (index != -1)

{

Angle longitude;

Angle latitude;

if ((longitude = AngleUI.TakeInputOfAngle("Longitude")) == null)

{

AngleUI.WrongAngle();

}

else if ((latitude = AngleUI.TakeInputOfAngle("Latitude")) == null)

{

AngleUI.WrongAngle();

}

else

{

ShipCRUD.ListOfShips[index].setLatitude(latitude);

ShipCRUD.ListOfShips[index].setLongitude(longitude);

}

ShipCRUD.WritingToFile();

}

else

ShipUI.ShipNotFound();

}

static void Main(string[] args)

{

ShipCRUD.ReadFromFile();

char option;

do

{

MenuUI.Header();

option = MenuUI.ShipMenu();

switch (option)

{

case '1':

option\_1();

break;

case '2':

option\_2();

break;

case '3':

option\_3();

break;

case '4':

option\_4();

break;

case '5':

MenuUI.Header();

MenuUI.PrintMsg("Thansks FOr Using Our Application....");

break;

}

MenuUI.InputMsg();

} while (option != '5');

}

}

**Sequence Diagram**

**Point Of Sale**

**Case Study**

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.

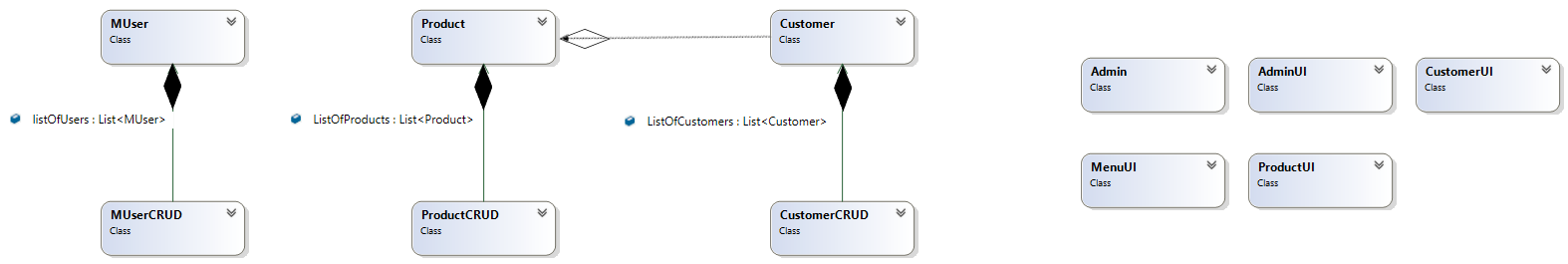
On All Grocery type of products, the sales tax is 10%, on all fruit types the tax is 5% and

if there is any other type the tax is 15%

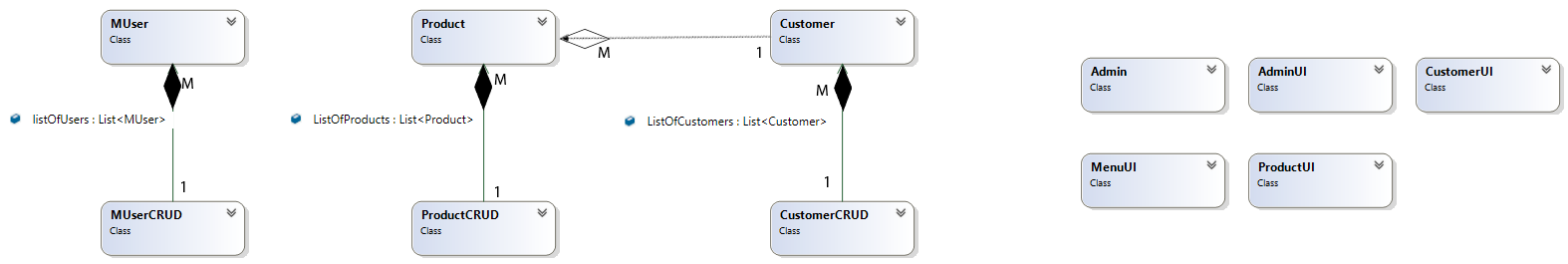
**Domain Model With Class Names**



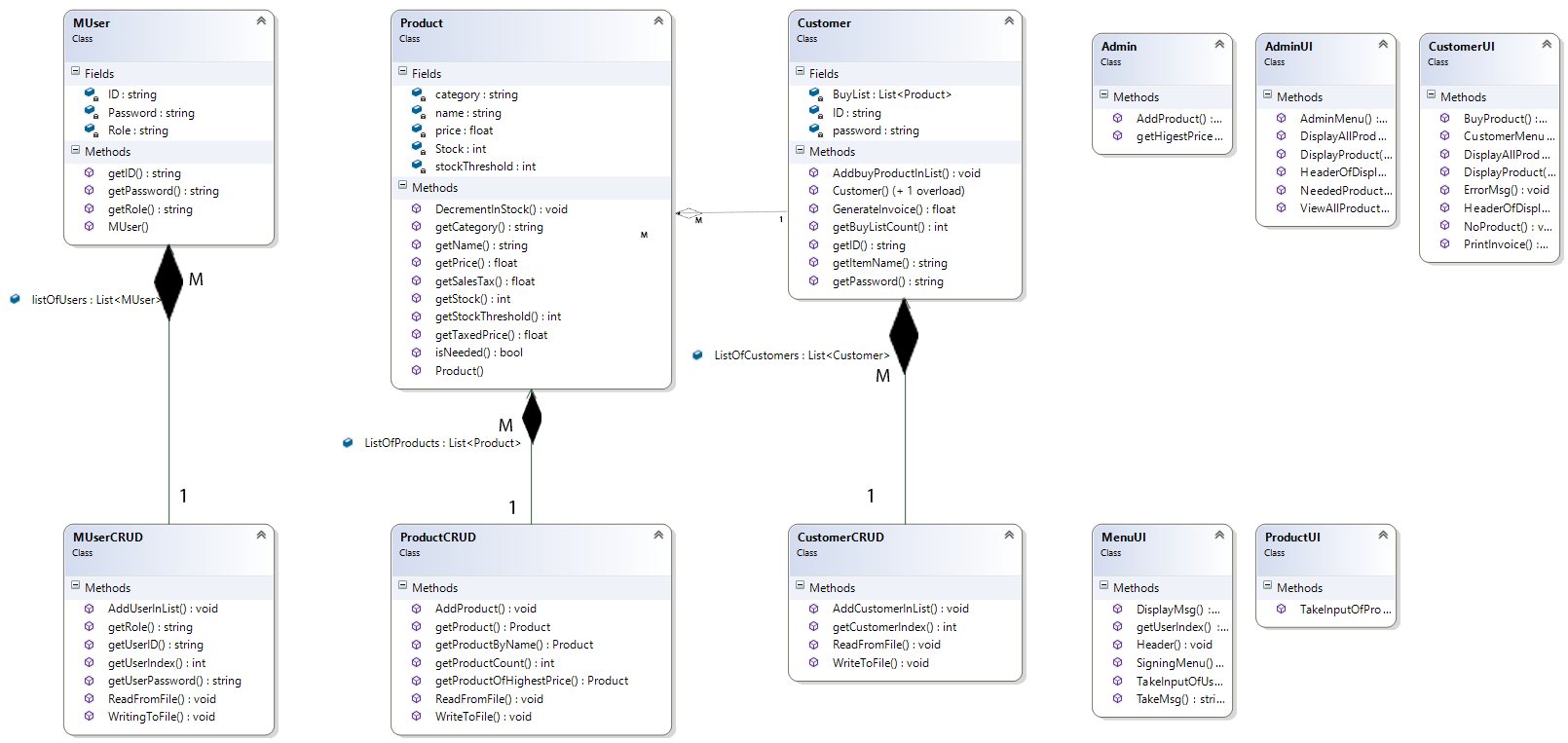
**Domain Model With Relations**



**Domain Model With Multiplicity**



**Class Diagram**



**Complete Code**

**BL:**

**Admin:**

class Admin

{

public void AddProduct(Product Source)

{

ProductCRUD.AddProduct(Source);

}

public Product getHigestPriceProduct()

{

return ProductCRUD.getProductOfHighestPrice();

}

}

**Customer:**

class Customer

{

public Customer(string ID, string password)

{

this.ID = ID;

this.password = password;

}

public Customer(string ID, string password, List<Product> BuyList)

{

this.ID = ID;

this.password = password;

this.BuyList = BuyList;

}

private string ID;

private string password;

private List<Product> BuyList = new List<Product>();

public void AddbuyProductInList(Product Source)

{

BuyList.Add(Source);

}

public string getID()

{

return ID;

}

public string getPassword()

{

return password;

}

public float GenerateInvoice()

{

float Invoice = 0;

for (int i = 0; i < BuyList.Count; i++)

{

Invoice = Invoice + BuyList[i].getTaxedPrice();

}

return Invoice;

}

public int getBuyListCount()

{

return BuyList.Count;

}

public string getItemName(int index)

{

return BuyList[index].getName();

}

}

**MUser:**

class MUser

{

public MUser(string ID, string Password, string Role)

{

this.ID = ID;

this.Password = Password;

this.Role = Role;

}

private string ID;

private string Password;

private string Role;

// public static List<> ListOfCustomers = new List<Customer>();

public string getID()

{

return ID;

}

public string getPassword()

{

return Password;

}

public string getRole()

{

return Role;

}

}

**Product**

class Product

{

public Product(string name, string category, float price, int stock, int stockThreshold)

{

this.name = name;

this.category = category;

this.price = price;

this.stockThreshold = stockThreshold;

this.Stock = stock;

}

private string name;

private string category;

private float price;

private int Stock;

private int stockThreshold;

public float getPrice()

{

return price;

}

public string getName()

{

return name;

}

public string getCategory()

{

return category;

}

public void DecrementInStock()

{

Stock--;

}

public float getTaxedPrice()

{

float Tax = 0;

if (category == "Grocery")

{

Tax = 10;

}

else if (category == "Fruit")

{

Tax = 5;

}

else

Tax = 15;

return price + ((price \* Tax) / 100);

}

public float getSalesTax()

{

return getTaxedPrice() - price;

}

public bool isNeeded()

{

if (Stock < stockThreshold)

return true;

else return false;

}

public int getStock()

{

return Stock;

}

public int getStockThreshold()

{

return stockThreshold;

}

}

**DL:**

**CustomerCRUD:**

class CustomerCRUD

{

public static List<Customer> ListOfCustomers = new List<Customer>();

public static int getCustomerIndex(string ID, string password)

{

for (int i = 0; i < ListOfCustomers.Count; i++)

{

if (ListOfCustomers[i].getID() == ID && ListOfCustomers[i].getPassword() == password)

return i;

}

return -1;

}

public static void AddCustomerInList(Customer source)

{

ListOfCustomers.Add(source);

}

public static void WriteToFile()

{

string Path = "Cust.txt";

StreamWriter File = new StreamWriter(Path);

for (int i = 0; i < ListOfCustomers.Count; i++)

{

File.Write(ListOfCustomers[i].getID() + "," + ListOfCustomers[i].getPassword() + ",");

if (ListOfCustomers[i].getBuyListCount() != 0)

{

for (int j = 0; j < ListOfCustomers[i].getBuyListCount() - 1; j++)

{

File.Write(ListOfCustomers[i].getItemName(j) + ";");

}

File.Write(ListOfCustomers[i].getItemName(ListOfCustomers[i].getBuyListCount() - 1));

}

else

{

File.Write("-1");

}

File.WriteLine();

File.Flush();

}

File.Close();

}

public static void ReadFromFile()

{

string path = "Cust.txt";

StreamReader File = new StreamReader(path);

string temp = " ";

while ((temp = File.ReadLine()) != null)

{

string[] fields = temp.Split(',');

if (fields[2] != "-1")

{

string[] field\_2 = fields[2].Split(';');

List<Product> buyList = new List<Product>();

for (int j = 0; j < field\_2.Length; j++)

{

Product Temp = ProductCRUD.getProductByName(field\_2[j]);

if (Temp != null)

buyList.Add(Temp);

}

ListOfCustomers.Add(new Customer(fields[0], fields[1], buyList));

}

else

{

ListOfCustomers.Add(new Customer(fields[0], fields[1]));

}

}

File.Close();

}

}

**MUSerCRUD:**

class MUserCRUD

{

public static List<MUser> listOfUsers = new List<MUser>();

public static void AddUserInList(MUser Source)

{

listOfUsers.Add(Source);

}

public static int getUserIndex(string ID, string Password)

{

for (int i = 0; i < listOfUsers.Count; i++)

{

if (ID == listOfUsers[i].getID() && Password == listOfUsers[i].getPassword())

return i;

}

return -1;

}

public static string getRole(int index)

{

return listOfUsers[index].getRole();

}

public static string getUserID(int index)

{

return listOfUsers[index].getID();

}

public static string getUserPassword(int index)

{

return listOfUsers[index].getPassword();

}

public static void ReadFromFile()

{

string path = "Users.txt";

StreamReader File = new StreamReader(path);

string temp = " ";

while ((temp = File.ReadLine()) != null)

{

string ID;

string Password;

string Role;

string[] fields = temp.Split(',');

ID = fields[0];

Password = fields[1];

Role = fields[2];

MUserCRUD.AddUserInList(new MUser(ID, Password, Role));

}

File.Close();

}

public static void WritingToFile()

{

string path = "Users.txt";

StreamWriter File = new StreamWriter(path);

for (int i = 0; i < listOfUsers.Count; i++)

{

File.WriteLine(listOfUsers[i].getID() + "," + listOfUsers[i].getPassword() + "," + listOfUsers[i].getRole());

File.Flush();

}

File.Close();

}

}

**ProductCRUD:**

class ProductCRUD

{

public static List<Product> ListOfProducts = new List<Product>();

public static void AddProduct(Product Source)

{

ListOfProducts.Add(Source);

}

public static Product getProduct(int index)

{

return ListOfProducts[index];

}

public static Product getProductByName(string name)

{

for (int i = 0; i < ListOfProducts.Count; i++)

{

if (ListOfProducts[i].getName() == name)

{

return ListOfProducts[i];

}

}

return null;

}

public static int getProductCount()

{

return ListOfProducts.Count;

}

public static Product getProductOfHighestPrice()

{

float HighestPrice;

int HighestPriceIndex = 0;

HighestPrice = ListOfProducts[0].getPrice();

for (int i = 1; i < ListOfProducts.Count; i++)

{

if (HighestPrice < ListOfProducts[i].getPrice())

{

HighestPriceIndex = i;

HighestPrice = ListOfProducts[i].getPrice();

}

}

return ListOfProducts[HighestPriceIndex];

}

public static void WriteToFile()

{

string Path = "Products.txt";

StreamWriter File = new StreamWriter(Path);

for (int i = 0; i < ListOfProducts.Count; i++)

{

File.WriteLine(ListOfProducts[i].getName() + "," + ListOfProducts[i].getCategory() + "," + ListOfProducts[i].getPrice() + "," + ListOfProducts[i].getStock() + "," + ListOfProducts[i].getStockThreshold());

File.Flush();

}

File.Close();

}

public static void ReadFromFile()

{

string path = "Products.txt";

StreamReader File = new StreamReader(path);

string temp = " ";

while ((temp = File.ReadLine()) != null)

{

string[] fields = temp.Split(',');

ProductCRUD.AddProduct(new Product(fields[0], fields[1], float.Parse(fields[2]), int.Parse(fields[3]), int.Parse(fields[4])));

}

File.Close();

}

}

**UI:**

**AdminUI:**

class AdminUI

{

public static char AdminMenu()

{

Console.WriteLine("Press 1. To Add Product");

Console.WriteLine("Press 2. To View All Products");

Console.WriteLine("Press 3. To Find Product With Highest Price");

Console.WriteLine("Press 4. To View Sales Tax Of All Products");

Console.WriteLine("Press 5. To Products To Be Ordered");

Console.WriteLine("Press 6. To Exit");

char option;

char.TryParse(Console.ReadLine(), out option);

return option;

}

public static void HeaderOfDisplayProduct()

{

int y = Console.CursorTop;

Console.Write("Name");

Console.SetCursorPosition(20, y);

Console.Write("Category");

Console.SetCursorPosition(40, y);

Console.WriteLine("Price");

}

public static void DisplayProduct(string name, string category, float price)

{

int y = Console.CursorTop;

Console.Write(name);

Console.SetCursorPosition(20, y);

Console.Write(category);

Console.SetCursorPosition(40, y);

Console.WriteLine(price);

}

public static void DisplayAllProducts()

{

Console.WriteLine(".....All Products.......");

HeaderOfDisplayProduct();

for (int i = 0; i < ProductCRUD.ListOfProducts.Count; i++)

{

string name = ProductCRUD.ListOfProducts[i].getName();

string category = ProductCRUD.ListOfProducts[i].getCategory();

float price = ProductCRUD.ListOfProducts[i].getPrice();

DisplayProduct(name, category, price);

}

}

public static void ViewAllProductsSalesTax()

{

Console.WriteLine(".......Sales tax of all Products.......");

for (int i = 0; i < ProductCRUD.ListOfProducts.Count; i++)

{

Console.WriteLine((i + 1).ToString() + ". " + ProductCRUD.ListOfProducts[i].getName() + " With Sales tax " + ProductCRUD.ListOfProducts[i].getSalesTax().ToString());

}

}

public static void NeededProducts()

{

bool flag = true;

for (int i = 0; i < ProductCRUD.ListOfProducts.Count; i++)

{

if (ProductCRUD.ListOfProducts[i].isNeeded())

{

Console.WriteLine(ProductCRUD.ListOfProducts[i].getName() + " is Needed to be Ordered...");

flag = false;

}

}

if (flag)

Console.WriteLine("Nothing to Order....");

}

}

**CustomerUI:**

class CustomerUI

{

public static char CustomerMenu()

{

Console.WriteLine("Press 1. To View All Products");

Console.WriteLine("Press 2. To Buy Products");

Console.WriteLine("Press 3. To Generate Invoice");

Console.WriteLine("Press 4. To Exit");

char option;

char.TryParse(Console.ReadLine(), out option);

return option;

}

public static void HeaderOfDisplayProduct()

{

int y = Console.CursorTop;

Console.Write("Name");

Console.SetCursorPosition(20, y);

Console.Write("Category");

Console.SetCursorPosition(40, y);

Console.WriteLine("Price");

}

public static void DisplayProduct(string name, string category, float price)

{

int y = Console.CursorTop;

Console.Write(name);

Console.SetCursorPosition(20, y);

Console.Write(category);

Console.SetCursorPosition(40, y);

Console.WriteLine(price);

}

public static void ErrorMsg()

{

Console.WriteLine("There is some Error in Database...");

Console.ReadLine();

}

public static void DisplayAllProducts()

{

Console.WriteLine(".....All Products.......");

HeaderOfDisplayProduct();

for (int i = 0; i < ProductCRUD.ListOfProducts.Count; i++)

{

string name = ProductCRUD.ListOfProducts[i].getName();

string category = ProductCRUD.ListOfProducts[i].getCategory();

float price = ProductCRUD.ListOfProducts[i].getPrice();

DisplayProduct(name, category, price);

}

}

public static void PrintInvoice(float Invoice)

{

Console.WriteLine("You Have To Pay = " + Invoice);

}

public static Product BuyProduct()

{

Console.WriteLine(" ");

Console.WriteLine("Which Product You want to buy");

string option = Console.ReadLine();

int index = int.Parse(option);

Product temp = ProductCRUD.getProduct(index - 1);

return temp;

}

public static void NoProduct()

{

Console.WriteLine("No Product Available....");

}

}

**MenuUI:**

class MenuUI

{

public static void Header()

{

Console.Clear();

Console.WriteLine("#####################################");

Console.WriteLine("# Point of Sale #");

Console.WriteLine("#####################################");

}

public static char SigningMenu()

{

Console.WriteLine("Press 1. To Sign In");

Console.WriteLine("Press 2. To Sign Up");

Console.WriteLine("Press 3. To Exit");

char option;

char.TryParse(Console.ReadLine(), out option);

return option;

}

public static MUser TakeInputOfUser()

{

string Role;

string ID;

string Password;

Console.WriteLine("ENter ID = ");

ID = Console.ReadLine();

Console.WriteLine("ENter Password = ");

Password = Console.ReadLine();

Console.WriteLine("ENter Role = ");

Role = Console.ReadLine();

MUser temp = new MUser(ID, Password, Role);

return temp;

}

public static void DisplayMsg(string Msg)

{

Console.WriteLine(Msg);

}

public static string TakeMsg()

{

return Console.ReadLine();

}

public static int getUserIndex()

{

string ID = "";

string Password = "";

MenuUI.DisplayMsg("Enter Your ID....");

ID = MenuUI.TakeMsg();

MenuUI.DisplayMsg("Enter Your Password....");

Password = MenuUI.TakeMsg();

int Userindex = MUserCRUD.getUserIndex(ID, Password);

return Userindex;

}

}

**ProductUI:**

class ProductUI

{

public static Product TakeInputOfProduct()

{

string name;

string category;

float price;

int stock;

int threshold;

Console.WriteLine("ENter Name Of Product...");

name = Console.ReadLine();

Console.WriteLine("Enter Category of Product...");

category = Console.ReadLine();

Console.WriteLine("Enter Price Of Product...");

price = float.Parse(Console.ReadLine());

Console.WriteLine("Enter Available Stock...");

stock = int.Parse(Console.ReadLine());

Console.WriteLine("Enter Stock Threshold...");

threshold = int.Parse(Console.ReadLine());

Product temp = new Product(name, category, price, stock, threshold);

return temp;

}

}

**Driver Program**

static void Cust\_Option\_1()

{

MenuUI.Header();

CustomerUI.DisplayAllProducts();

}

static void Cust\_Option\_2(int CustNo)

{

MenuUI.Header();

if (ProductCRUD.getProductCount() == 0)

{

CustomerUI.NoProduct();

MenuUI.TakeMsg();

}

else

{

CustomerUI.DisplayAllProducts();

Product ChossenProduct = CustomerUI.BuyProduct();

ChossenProduct.DecrementInStock();

CustomerCRUD.ListOfCustomers[CustNo].AddbuyProductInList(ChossenProduct);

ProductCRUD.WriteToFile();

CustomerCRUD.WriteToFile();

}

}

static void Cust\_Option\_3(int CustNo)

{

MenuUI.Header();

CustomerUI.PrintInvoice(CustomerCRUD.ListOfCustomers[CustNo].GenerateInvoice());

}

static void Admin\_Option\_1(Admin ActorAdmin)

{

MenuUI.Header();

ActorAdmin.AddProduct(ProductUI.TakeInputOfProduct());

ProductCRUD.WriteToFile();

}

static void Admin\_Option\_2()

{

MenuUI.Header();

AdminUI.DisplayAllProducts();

}

static void Admin\_Option\_3(Admin ActorAdmin)

{

MenuUI.Header();

Product temp = ActorAdmin.getHigestPriceProduct();

AdminUI.HeaderOfDisplayProduct();

AdminUI.DisplayProduct(temp.getName(), temp.getCategory(), temp.getPrice());

}

static void Admin\_Option\_4()

{

MenuUI.Header();

AdminUI.ViewAllProductsSalesTax();

}

static void Admin\_Option\_5()

{

MenuUI.Header();

AdminUI.NeededProducts();

}

static void Main(string[] args)

{

ProductCRUD.ReadFromFile();

CustomerCRUD.ReadFromFile();

MUserCRUD.ReadFromFile();

char UserSelectionOption;

do

{

MenuUI.Header();

UserSelectionOption = MenuUI.SigningMenu();

if (UserSelectionOption == '1')

{

MenuUI.Header();

int Userindex = MenuUI.getUserIndex();

if (Userindex != -1)

{

string WHO;

WHO = MUserCRUD.getRole(Userindex);

if (WHO == "Cust")

{

int CustNo = CustomerCRUD.getCustomerIndex(MUserCRUD.getUserID(Userindex), MUserCRUD.getUserPassword(Userindex));

if (CustNo != -1)

{

char CustOption;

do

{

MenuUI.Header();

CustOption = CustomerUI.CustomerMenu();

if (CustOption == '1')

{

Cust\_Option\_1();

}

else if (CustOption == '2')

{

Cust\_Option\_2(CustNo);

}

else if (CustOption == '3')

{

Cust\_Option\_3(CustNo);

}

Console.ReadKey();

} while (CustOption != '4');

}

else

{

CustomerUI.ErrorMsg();

}

}

else if (WHO == "Admin")

{

Admin ActorAdmin = new Admin();

char AdminOption;

do

{

MenuUI.Header();

AdminOption = AdminUI.AdminMenu();

if (AdminOption == '1')

{

Admin\_Option\_1(ActorAdmin);

}

else if (AdminOption == '2')

{

Admin\_Option\_2();

}

else if (AdminOption == '3')

{

Admin\_Option\_3(ActorAdmin);

}

else if (AdminOption == '4')

{

Admin\_Option\_4();

}

else if (AdminOption == '5')

{

Admin\_Option\_5();

}

Console.ReadKey();

} while (AdminOption != '6');

}

}

else

MenuUI.DisplayMsg("....Wrong Credentials...");

}

else if (UserSelectionOption == '2')

{

MUser Input = MenuUI.TakeInputOfUser();

if (Input.getRole() == "Cust")

{

CustomerCRUD.AddCustomerInList(new Customer(Input.getID(), Input.getPassword()));

CustomerCRUD.WriteToFile();

}

MUserCRUD.AddUserInList(Input);

MUserCRUD.WritingToFile();

}

Console.ReadKey();

} while (UserSelectionOption != '3');

MUserCRUD.WritingToFile();

}

}

**Sequence Diagram**

**Point On Line**

**Case Study**

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

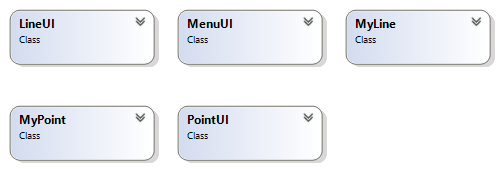
point at (x2, y2), is designed as shown in the class diagram. The MyLine class uses two

MyPoint instances (written in the earlier exercise) as its begin and end points. Write the

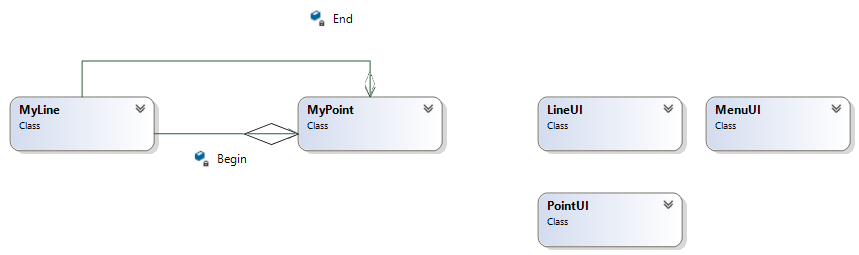
MyLine class. Also write a test driver to test all the public methods in the MyLine class.

Use distance formula to calculate the length of the line

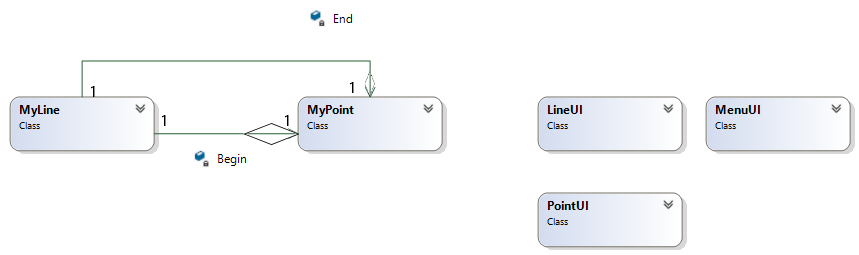
**Domain Model with Class Names**



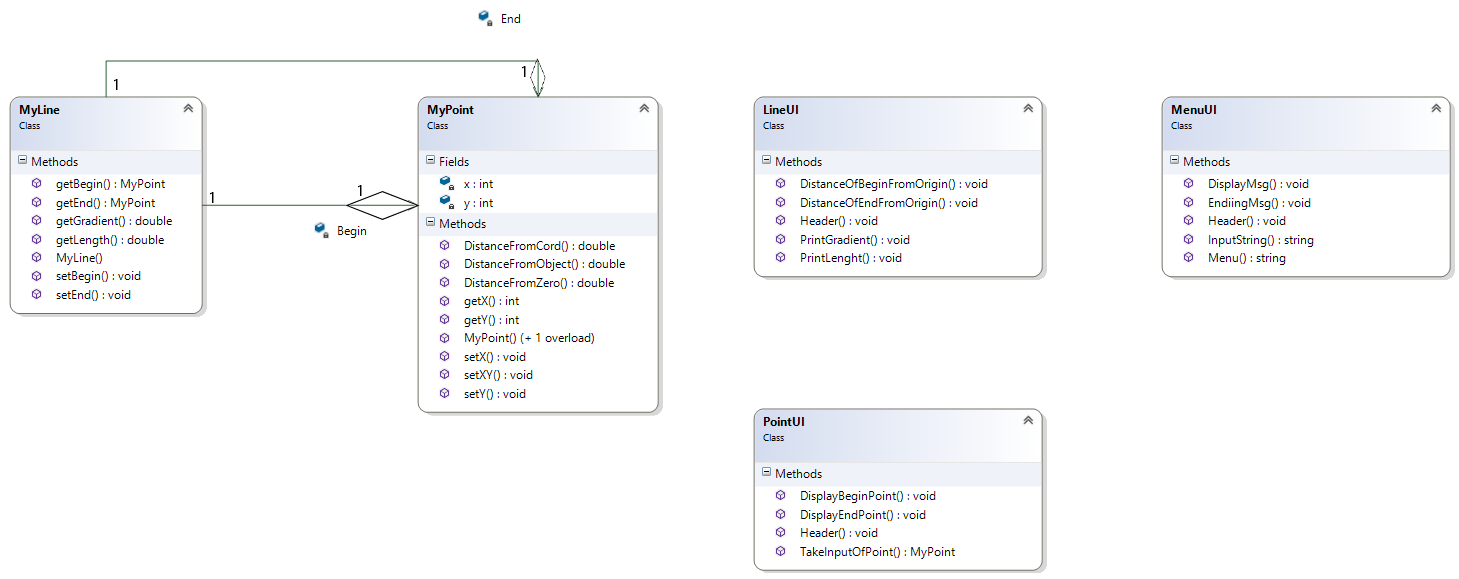
**Domain Model With Relations**



**Domain Model With Multiplicity**



**Class Diagram**



**Complete Code**

**BL:**

**MyLine:**

class MyLine

{

public MyLine(MyPoint Begin, MyPoint End)

{

this.Begin = Begin;

this.End = End;

}

private MyPoint Begin;

private MyPoint End;

public MyPoint getBegin()

{

return Begin;

}

public MyPoint getEnd()

{

return End;

}

public void setBegin(MyPoint Begin)

{

this.Begin = Begin;

}

public void setEnd(MyPoint End)

{

this.End = End;

}

public double getLength()

{

double length = Begin.DistanceFromObject(End);

return length;

}

public double getGradient()

{

double m;

m = (End.getY() - Begin.getY()) / (End.getX() - Begin.getX());

return m;

}

}

**MyPoint:**

class MyPoint

{

public MyPoint()

{

}

public MyPoint(int x, int y)

{

this.x = x;

this.y = y;

}

private int x;

private int y;

public int getX()

{

return x;

}

public int getY()

{

return y;

}

public void setX(int x)

{

this.x = x;

}

public void setY(int y)

{

this.y = y;

}

public void setXY(int x, int y)

{

this.x = x;

this.y = y;

}

public double DistanceFromCord(int x, int y)

{

double distance = Math.Sqrt(Math.Pow((this.x - x), 2) + Math.Pow((this.y - y), 2));

return distance;

}

public double DistanceFromObject(MyPoint obj)

{

double distance = Math.Sqrt(Math.Pow((this.x - obj.getX()), 2) + Math.Pow((this.y - obj.getY()), 2));

return distance;

}

public double DistanceFromZero()

{

double distance = Math.Sqrt(Math.Pow((this.x - 0), 2) + Math.Pow((this.y - 0), 2));

return distance;

}

}

**UI:**

**LineUI:**

class LineUI

{

public static void Header()

{

Console.Clear();

Console.WriteLine("###############################################");

Console.WriteLine("# Point on Line Application #");

Console.WriteLine("###############################################");

}

public static void PrintLenght(double lenght)

{

Console.WriteLine(".........Length of Line........");

Console.WriteLine("Length = " + lenght);

}

public static void PrintGradient(double gradient)

{

Console.WriteLine(".........Gradient of Line........");

Console.WriteLine("Gradient of Line = " + gradient);

}

public static void DistanceOfBeginFromOrigin(double distance)

{

Console.WriteLine("........Diatance of begin From Origin.........");

Console.WriteLine("Distance = " + distance);

}

public static void DistanceOfEndFromOrigin(double distance)

{

Console.WriteLine("........Diatance of End From Origin.........");

Console.WriteLine("Distance = " + distance);

}

}

**MenuUI:**

class MenuUI

{

public static void DisplayMsg(string msg)

{

Console.WriteLine(msg);

}

public static string InputString()

{

return Console.ReadLine();

}

public static void Header()

{

Console.Clear();

Console.WriteLine("###############################################");

Console.WriteLine("# Point on Line Application #");

Console.WriteLine("###############################################");

}

public static string Menu()

{

Console.WriteLine("Press 1. To Make a Line");

Console.WriteLine("Press 2. Update The Begin Point");

Console.WriteLine("Press 3. Update the End Point");

Console.WriteLine("Press 4. Show the begin Point");

Console.WriteLine("Press 5. Show The End Point");

Console.WriteLine("Press 6. Get The Lenght of the Line");

Console.WriteLine("Press 7. Get The Gradient of The Line");

Console.WriteLine("Press 8. Find Distance of the Begin Point From Origin");

Console.WriteLine("Press 9. Find Distance of the End Point From Origin");

Console.WriteLine("Press 10. Exit");

string option;

option = Console.ReadLine();

return option;

}

public static void EndiingMsg()

{

Console.WriteLine("............Thanks For Using our Application........");

}

}

**PointUI:**

class PointUI

{

public static void Header()

{

Console.Clear();

Console.WriteLine("###############################################");

Console.WriteLine("# Point on Line Application #");

Console.WriteLine("###############################################");

}

public static MyPoint TakeInputOfPoint(string type)

{

Console.WriteLine(".........Enter " + type + " Point......");

Console.WriteLine("ENter Value of abcissa :");

int x;

x = int.Parse(Console.ReadLine());

Console.WriteLine("Enter Value of Ordinate : ");

int y;

y = int.Parse(Console.ReadLine());

MyPoint temp = new MyPoint(x, y);

return temp;

}

public static void DisplayBeginPoint(MyPoint P)

{

Console.WriteLine("Begin Point = ( " + P.getX() + " , " + P.getY() + " )");

}

public static void DisplayEndPoint(MyPoint P)

{

Console.WriteLine("End Point = ( " + P.getX() + " , " + P.getY() + " )");

}

}

**Driver Program:**

class Program

{

static MyLine option\_1()

{

MenuUI.Header();

MyPoint Begin = PointUI.TakeInputOfPoint("Begin");

MyPoint End = PointUI.TakeInputOfPoint("End");

MyLine Line = new MyLine(Begin, End);

MenuUI.InputString();

return Line;

}

static void option\_2(MyLine Line)

{

MenuUI.Header();

MenuUI.DisplayMsg("......Update Begin Point.....");

Line.setBegin(PointUI.TakeInputOfPoint("Begin"));

MenuUI.InputString();

}

static void option\_3(MyLine Line)

{

MenuUI.Header();

MenuUI.DisplayMsg("......Update End Point.....");

Line.setBegin(PointUI.TakeInputOfPoint("End"));

MenuUI.InputString();

}

static void option\_4(MyLine Line)

{

MenuUI.Header();

PointUI.DisplayBeginPoint(Line.getBegin());

MenuUI.InputString();

}

static void option\_5(MyLine Line)

{

MenuUI.Header();

PointUI.DisplayBeginPoint(Line.getEnd());

MenuUI.InputString();

}

static void option\_6(MyLine Line)

{

MenuUI.Header();

LineUI.PrintLenght(Line.getLength());

MenuUI.InputString();

}

static void option\_7(MyLine Line)

{

MenuUI.Header();

LineUI.PrintGradient(Line.getGradient());

MenuUI.InputString();

}

static void option\_8(MyLine Line)

{

MenuUI.Header();

LineUI.DistanceOfBeginFromOrigin(Line.getBegin().DistanceFromZero());

MenuUI.InputString();

}

static void option\_9(MyLine Line)

{

MenuUI.Header();

LineUI.DistanceOfEndFromOrigin(Line.getEnd().DistanceFromZero());

MenuUI.InputString();

}

static void Main(string[] args)

{

MyLine Line = null;

string option;

do

{

MenuUI.Header();

option = MenuUI.Menu();

switch (option)

{

case "1":

Line = option\_1();

break;

case "2":

option\_2(Line);

break;

case "3":

option\_3(Line);

break;

case "4":

option\_4(Line);

break;

case "5":

option\_5(Line);

break;

case "6":

option\_6(Line);

break;

case "7":

option\_7(Line);

break;

case "8":

option\_8(Line);

break;

case "9":

option\_9(Line);

break;

case "10":

MenuUI.Header();

MenuUI.EndiingMsg();

break;

default:

MenuUI.Header();

MenuUI.DisplayMsg("........Wrong Opiton.....");

break;

}

MenuUI.InputString();

} while (option != "10");

}

}

}

**Sequence Diagram**

**UAMS**

**Case Study**

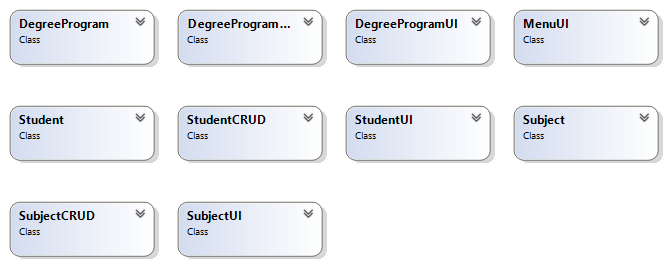
Acedemic branch offers different programs within different departments each program has a degree title and duration of degree.

Student apply for admisision in university and provides his/her name , age , FSC , And Ecat Marks and selects any number of preferences among the available programs.

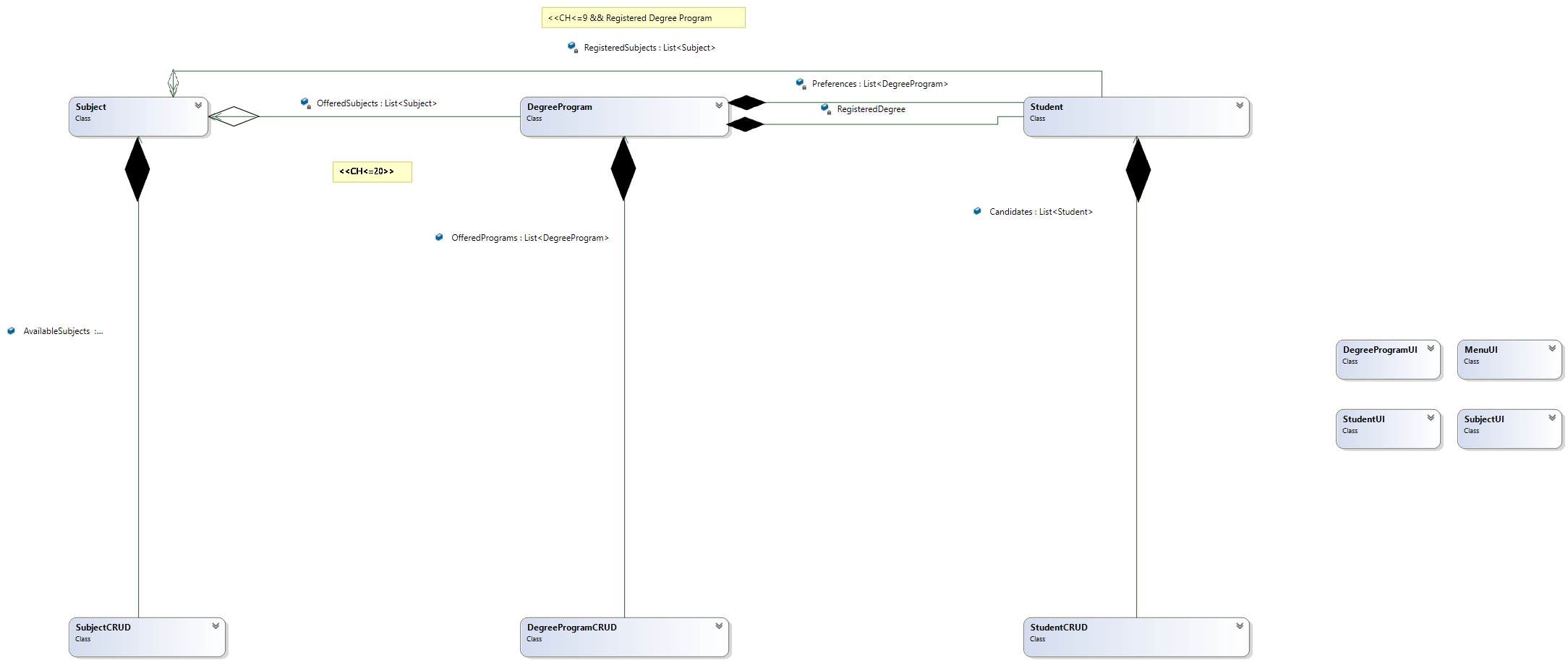
Admission branch prepares a merit list according to the highest merit and acedemeic branch also add subjects for each program . a subject have subject code , credit hours , subject type. A program cannot have more than 20 Credit hour subjects. A student registers multiple subjects but he/;she cannot take more than 9 credit hours.

Fee department generate fees according to registered subjects of the students.

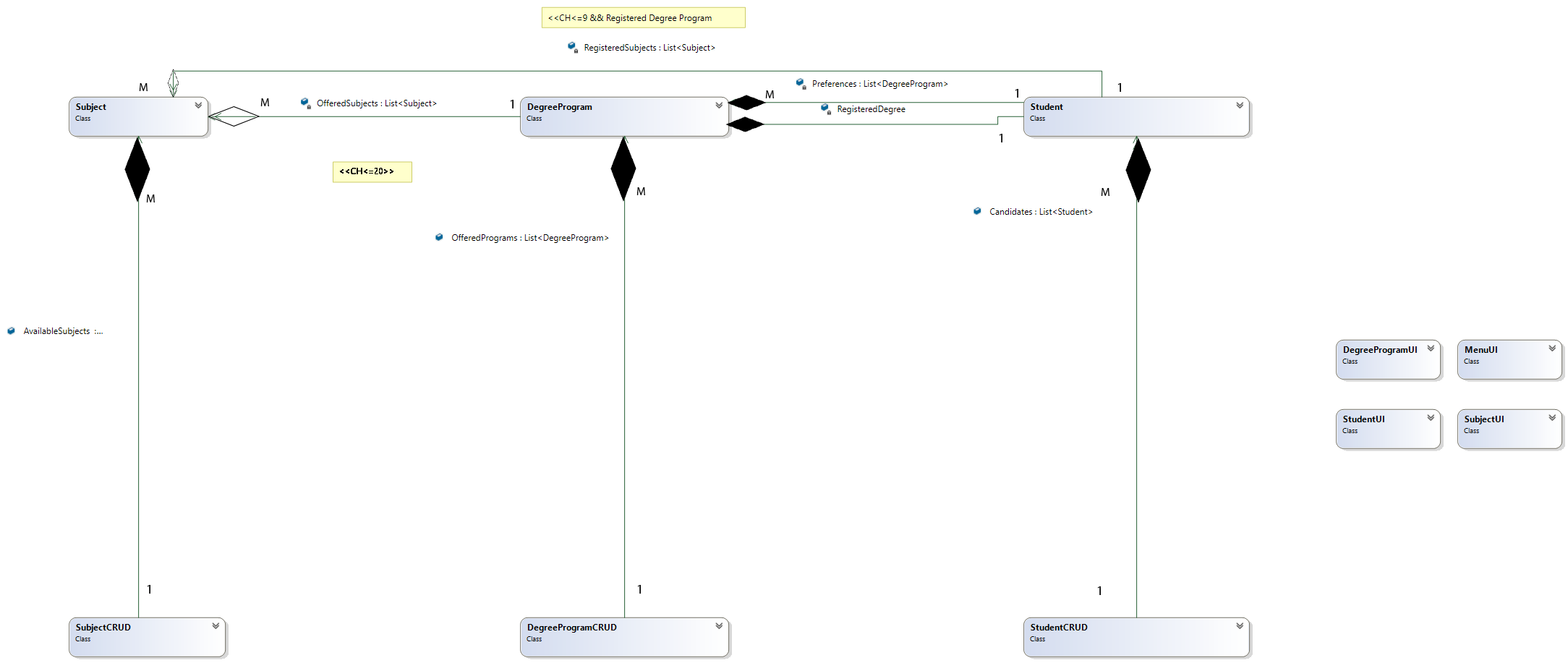
**Domain Model With Class Names**



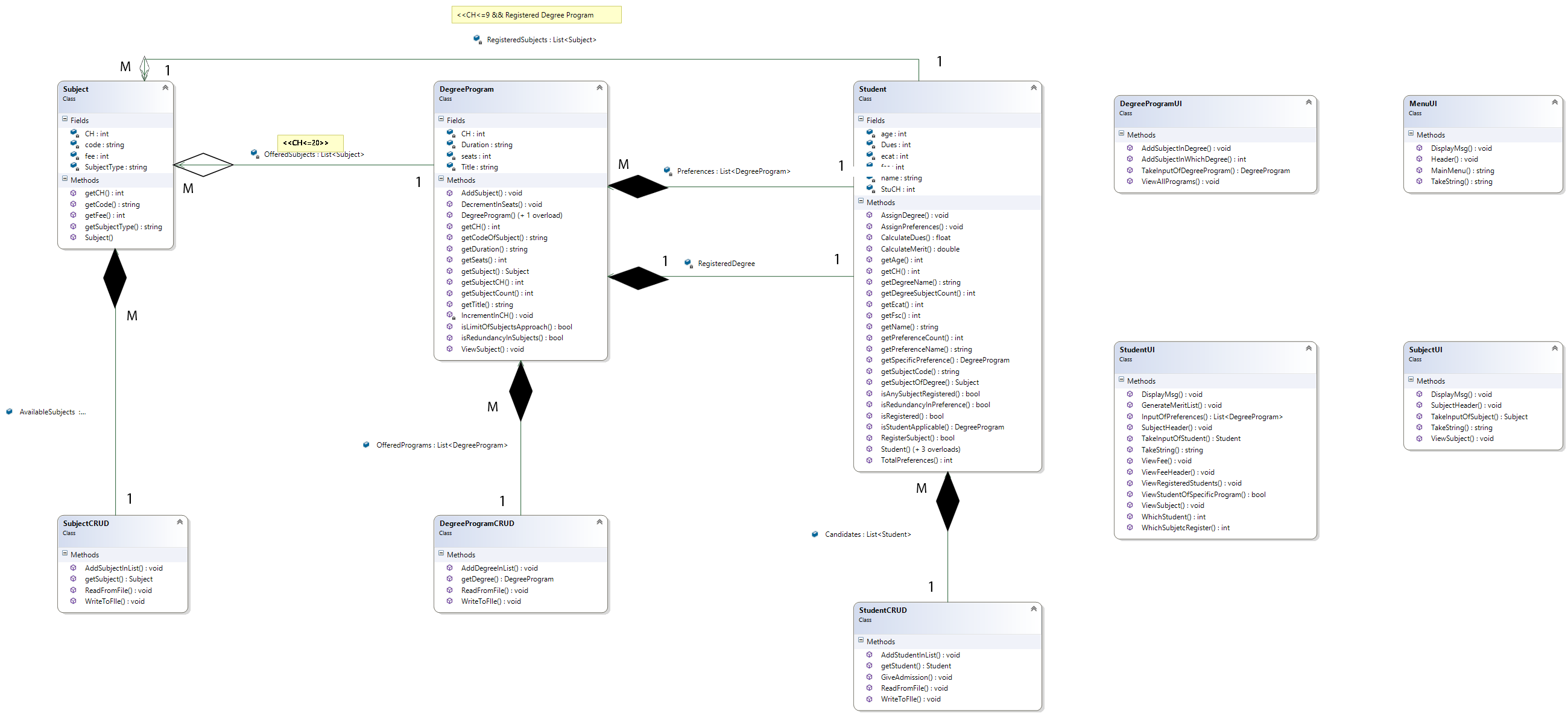
**Domain Model with Relations**



**Domain Model with Multiplicity**



**Class Diagram**



**Complete Code**

**BL:**

**Degree Program:**

class DegreeProgram

{

public DegreeProgram(string Title, string Duration, int seats)

{

this.Title = Title;

this.Duration = Duration;

this.seats = seats;

}

public DegreeProgram(string Title, string Duration, int seats, int CH)

{

this.Title = Title;

this.Duration = Duration;

this.seats = seats;

this.CH = CH;

}

private List<Subject> OfferedSubjects = new List<Subject>();

private string Title;

private string Duration;

private int seats;

private int CH = 0;

public string getTitle()

{

return Title;

}

public string getDuration()

{

return Duration;

}

public int getSeats()

{

return seats;

}

public void DecrementInSeats()

{

seats--;

}

public void AddSubject(Subject Source)

{

if (CH <= 20)

{

OfferedSubjects.Add(Source);

IncrementInCH(Source.getCH());

}

else

MenuUI.DisplayMsg("Cannnot add more subjects");

}

private void IncrementInCH(int value)

{

CH = CH + value;

}

public int getCH()

{

return CH;

}

public int getSubjectCH(int index)

{

return OfferedSubjects[index - 1].getCH();

}

public void ViewSubject(int index)

{

MenuUI.DisplayMsg(OfferedSubjects[index - 1].getCode() + "\t" + OfferedSubjects[index - 1].getCH());

}

public int getSubjectCount()

{

return OfferedSubjects.Count;

}

public bool isLimitOfSubjectsApproach(int SubjectCH)

{

if ((SubjectCH + CH) > 20)

return true;

return false;

}

public bool isRedundancyInSubjects(Subject Source)

{

for (int i = 0; i < OfferedSubjects.Count; i++)

{

if (OfferedSubjects[i] == Source)

return true;

}

return false;

}

public Subject getSubject(int index)

{

return OfferedSubjects[index];

}

public string getCodeOfSubject(int index)

{

return OfferedSubjects[index].getCode();

}

}

**Student:**

class Student

{

public Student(string name, int age, int fsc, int ecat)

{

this.name = name;

this.age = age;

this.fsc = fsc;

this.ecat = ecat;

RegisteredSubjects = null;

}

public Student(string name, int age, int fsc, int ecat, int StuCH, List<DegreeProgram> Preferences, DegreeProgram RegisteredDegree, List<Subject> RegisteredSubjects)

{

this.name = name;

this.age = age;

this.fsc = fsc;

this.ecat = ecat;

this.StuCH = StuCH;

this.RegisteredDegree = RegisteredDegree;

this.RegisteredSubjects = RegisteredSubjects;

this.Preferences = Preferences;

}

public Student(string name, int age, int fsc, int ecat, int StuCH, List<DegreeProgram> Preferences, DegreeProgram RegisteredDegree)

{

this.name = name;

this.age = age;

this.fsc = fsc;

this.ecat = ecat;

this.StuCH = StuCH;

this.RegisteredDegree = RegisteredDegree;

this.Preferences = Preferences;

}

public Student(string name, int age, int fsc, int ecat, int StuCH, List<DegreeProgram> Preferences)

{

this.name = name;

this.age = age;

this.fsc = fsc;

this.ecat = ecat;

this.StuCH = StuCH;

this.Preferences = Preferences;

}

private string name;

private int fsc;

private int age;

private int ecat;

private List<DegreeProgram> Preferences = new List<DegreeProgram>();

private DegreeProgram RegisteredDegree = null;

private List<Subject> RegisteredSubjects = new List<Subject>();

private int StuCH;

private int Dues;

public double CalculateMerit()

{

double Merit = ((float)fsc / 1100 \* 70) + ((float)ecat / 400 \* 30);

return Merit;

}

public string getName()

{

return name;

}

public DegreeProgram getSpecificPreference(int prefNum)

{

return Preferences[prefNum];

}

public int TotalPreferences()

{

return Preferences.Count;

}

public void AssignPreferences(List<DegreeProgram> prefereces)

{

this.Preferences = prefereces;

}

public void AssignDegree(DegreeProgram source)

{

this.RegisteredDegree = source;

}

public bool isRegistered()

{

if (RegisteredDegree != null)

return true;

return false;

}

public int getDegreeSubjectCount()

{

if (RegisteredDegree == null)

return 0;

return RegisteredDegree.getSubjectCount();

}

public Subject getSubjectOfDegree(int index)

{

return RegisteredDegree.getSubject(index);

}

public bool RegisterSubject(Subject Source)

{

if (StuCH + Source.getCH() <= 9)

{

RegisteredSubjects.Add(Source);

StuCH = StuCH + Source.getCH();

return true;

}

return false;

}

public string getDegreeName()

{

if (RegisteredDegree == null)

return null;

return RegisteredDegree.getTitle();

}

public int getFsc()

{

return fsc;

}

public int getEcat()

{

return ecat;

}

public float CalculateDues()

{

Dues = 0;

for (int i = 0; i < RegisteredSubjects.Count; i++)

{

Dues = Dues + RegisteredSubjects[i].getFee();

}

return Dues;

}

public bool isRedundancyInPreference(List<DegreeProgram> Pref, DegreeProgram Source)

{

for (int i = 0; i < Pref.Count; i++)

{

Console.ReadLine();

if (Pref[i].getTitle() == Source.getTitle())

return true;

}

return false;

}

public DegreeProgram isStudentApplicable()

{

for (int i = 0; i < Preferences.Count; i++)

{

DegreeProgram PrefOfStudent = getSpecificPreference(i);

if (PrefOfStudent.getSeats() != 0)

{

PrefOfStudent.DecrementInSeats();

return PrefOfStudent;

}

}

return null;

}

public int getAge()

{

return age;

}

public int getCH()

{

return StuCH;

}

public string getSubjectCode(int index)

{

return RegisteredSubjects[index].getCode();

}

public int getPreferenceCount()

{

return Preferences.Count;

}

public bool isAnySubjectRegistered()

{

if (RegisteredSubjects.Count != 0)

return true;

return false;

}

public string getPreferenceName(int index)

{

return Preferences[index].getTitle();

}

}

**Subject:**

class Subject

{

public Subject(string code, int CH, string SubjectType, int fee)

{

this.code = code;

this.CH = CH;

this.SubjectType = SubjectType;

this.fee = fee;

}

private string code;

private int CH;

private string SubjectType;

private int fee;

public int getCH()

{

return CH;

}

public string getCode()

{

return code;

}

public int getFee()

{

return fee;

}

public string getSubjectType()

{

return SubjectType;

}

}

**DL:**

**DegreeProgramCRUD:**

class DegreeProgramCRUD

{

public static List<DegreeProgram> OfferedPrograms = new List<DegreeProgram>();

public static void AddDegreeInList(DegreeProgram Source)

{

OfferedPrograms.Add(Source);

}

public static DegreeProgram getDegree(string title)

{

for (int i = 0; i < OfferedPrograms.Count; i++)

{

if (OfferedPrograms[i].getTitle() == title)

return OfferedPrograms[i];

}

return null;

}

public static void ReadFromFile()

{

string path = "DegreeProgram.txt";

StreamReader File = new StreamReader(path);

string temp = "";

while ((temp = File.ReadLine()) != null)

{

string Source = "";

Source = Source + temp;

string[] SeparatedFields = Source.Split(',');

/\* Console.WriteLine(SeparatedFields[3]);

Console.ReadLine();\*/

DegreeProgram degree = new DegreeProgram(SeparatedFields[0], SeparatedFields[1], int.Parse(SeparatedFields[2]), int.Parse(SeparatedFields[3]));

string[] separatedSubjects = SeparatedFields[4].Split(';');

for (int i = 0; i < separatedSubjects.Length; i++)

{

Subject sub;

if ((sub = SubjectCRUD.getSubject(separatedSubjects[i])) != null)

degree.AddSubject(sub);

}

AddDegreeInList(degree);

}

File.Close();

}

public static void WriteToFlle()

{

string path = "DegreeProgram.txt";

StreamWriter File = new StreamWriter(path, true);

for (int i = 0; i < OfferedPrograms.Count; i++)

{

File.Write(OfferedPrograms[i].getTitle() + "," + OfferedPrograms[i].getDuration() + "," + OfferedPrograms[i].getSeats() + "," + OfferedPrograms[i].getCH() + ",");

for (int j = 0; j < OfferedPrograms[i].getSubjectCount() - 1; j++)

{

File.Write(OfferedPrograms[i].getCodeOfSubject(j) + ";");

}

File.WriteLine(OfferedPrograms[i].getCodeOfSubject(OfferedPrograms[i].getSubjectCount() - 1));

File.Flush();

}

File.Close();

}

}

**StudentCRUD:**

class StudentCRUD

{

public static List<Student> Candidates = new List<Student>();

public static void AddStudentInList(Student Source)

{

Candidates.Add(Source);

}

public static void GiveAdmission()

{

List<Student> orderedList = new List<Student>();

orderedList = StudentCRUD.Candidates.OrderByDescending(o => o.CalculateMerit()).ToList();

List<Student> AdmittedStudents = new List<Student>();

for (int i = 0; i < orderedList.Count; i++)

{

DegreeProgram temp;

if ((temp = orderedList[i].isStudentApplicable()) != null)

{

StudentCRUD.Candidates[i].AssignDegree(temp);

}

}

}

public static Student getStudent(int index)

{

return Candidates[index];

}

public static void ReadFromFile()

{

string path = "Student.txt";

StreamReader File = new StreamReader(path);

string CurrentLineInFile = "";

while ((CurrentLineInFile = File.ReadLine()) != null)

{

string Source = "";

Source = Source + CurrentLineInFile;

string[] SeparatedFieldsByCommas = Source.Split(',');

string[] SubjectsFields = SeparatedFieldsByCommas[7].Split(':');

string[] PreferencesFields = SeparatedFieldsByCommas[5].Split(';');

List<Subject> ListOfSubjects = new List<Subject>();

List<DegreeProgram> ListOfPreferences = new List<DegreeProgram>();

DegreeProgram Degree;

for (int i = 0; i < PreferencesFields.Length; i++)

{

DegreeProgram Pref = DegreeProgramCRUD.getDegree(PreferencesFields[i]);

if (Pref != null)

ListOfPreferences.Add(Pref);

}

if (SeparatedFieldsByCommas[6] != "-1")

{

Degree = DegreeProgramCRUD.getDegree(SeparatedFieldsByCommas[6]);//return degree with this name

if (SeparatedFieldsByCommas[7] != "-1")

{

for (int j = 0; j < SubjectsFields.Length; j++)

{

Subject temp1 = SubjectCRUD.getSubject(SubjectsFields[j]);

if (temp1 != null)

ListOfSubjects.Add(temp1);

}

}

else

{

ListOfSubjects = null;

}

}

else

{

Degree = null;

}

Student Stu = null;

if (SeparatedFieldsByCommas[7] == "-1" && SeparatedFieldsByCommas[6] != "-1")

Stu = new Student(SeparatedFieldsByCommas[0], int.Parse(SeparatedFieldsByCommas[1]), int.Parse(SeparatedFieldsByCommas[2]), int.Parse(SeparatedFieldsByCommas[3]), int.Parse(SeparatedFieldsByCommas[4]), ListOfPreferences, Degree);

else if (SeparatedFieldsByCommas[7] == "-1" && SeparatedFieldsByCommas[6] == "-1")

Stu = new Student(SeparatedFieldsByCommas[0], int.Parse(SeparatedFieldsByCommas[1]), int.Parse(SeparatedFieldsByCommas[2]), int.Parse(SeparatedFieldsByCommas[3]), int.Parse(SeparatedFieldsByCommas[4]), ListOfPreferences);

else if (SeparatedFieldsByCommas[7] != "-1" && SeparatedFieldsByCommas[6] != "-1")

Stu = new Student(SeparatedFieldsByCommas[0], int.Parse(SeparatedFieldsByCommas[1]), int.Parse(SeparatedFieldsByCommas[2]), int.Parse(SeparatedFieldsByCommas[3]), int.Parse(SeparatedFieldsByCommas[4]), ListOfPreferences, Degree, ListOfSubjects);

AddStudentInList(Stu);

}

File.Close();

}

public static void WriteToFlle()

{

string path = "Student.txt";

StreamWriter File = new StreamWriter(path);

for (int i = 0; i < Candidates.Count; i++)

{

File.Write(Candidates[i].getName() + "," + Candidates[i].getAge() + "," + Candidates[i].getFsc() + "," + Candidates[i].getEcat() + "," + Candidates[i].getCH() + ",");

for (int j = 0; j < StudentCRUD.Candidates[i].getPreferenceCount() - 1; j++)

{

File.Write(StudentCRUD.Candidates[i].getPreferenceName(j) + ";");

}

File.Write(StudentCRUD.Candidates[i].getPreferenceName(StudentCRUD.Candidates[i].getPreferenceCount() - 1) + ",");//Last Preference

if (Candidates[i].isRegistered())

{

File.Write(StudentCRUD.Candidates[i].getDegreeName() + ",");

if (StudentCRUD.Candidates[i].isAnySubjectRegistered())// cheking if subjects are registered

{

for (int j = 0; j < StudentCRUD.Candidates[i].getDegreeSubjectCount() - 1; j++)

{

File.Write(StudentCRUD.Candidates[i].getSubjectCode(j) + ";");

}

File.Write(StudentCRUD.Candidates[i].getSubjectCode(StudentCRUD.Candidates[i].getDegreeSubjectCount() - 1));

}

else

{

File.Write("-1");

}

}

else

{

File.Write("-1,-1");

}

File.WriteLine();

File.Flush();

}

File.Close();

}

}

**SubjectCRUD:**

class SubjectCRUD

{

public static List<Subject> AvailableSubjects = new List<Subject>();

public static void AddSubjectInList(Subject Source)

{

AvailableSubjects.Add(Source);

}

public static void ReadFromFile()

{

string path = "Subjects.txt";

StreamReader File = new StreamReader(path);

string temp = "";

while ((temp = File.ReadLine()) != null)

{

string Source = "";

Source = Source + temp;

string[] SeparatedFields = Source.Split(',');

Subject sub = new Subject(SeparatedFields[0], int.Parse(SeparatedFields[1]), SeparatedFields[2], int.Parse(SeparatedFields[3]));

AddSubjectInList(sub);

}

File.Close();

}

public static void WriteToFlle()

{

string path = "Subjects.txt";

StreamWriter File = new StreamWriter(path);

for (int i = 0; i < AvailableSubjects.Count; i++)

{

File.WriteLine(AvailableSubjects[i].getCode() + "," + AvailableSubjects[i].getCH() + "," + AvailableSubjects[i].getSubjectType() + "," + AvailableSubjects[i].getFee());

File.Flush();

}

File.Close();

}

public static Subject getSubject(string code)

{

for (int i = 0; i < AvailableSubjects.Count; i++)

{

if (code == AvailableSubjects[i].getCode())

return AvailableSubjects[i];

}

return null;

}

}

**UI;**

**DegreeProgramUI:**

class DegreeProgramUI

{

public static int AddSubjectInWhichDegree()

{

char option;

Console.WriteLine("Select Subject....");

char.TryParse(Console.ReadLine(), out option);

return int.Parse(option.ToString());

}

public static void ViewAllPrograms()

{

int Y = Console.CursorTop;

Console.SetCursorPosition(5, Y);

Console.Write("Name");

Console.SetCursorPosition(30, Y);

Console.WriteLine("Duration");

for (int i = 0; i < DegreeProgramCRUD.OfferedPrograms.Count; i++)

{

Y = Console.CursorTop;

Console.SetCursorPosition(5, Y);

Console.Write(i + 1 + ". ");

Console.Write(DegreeProgramCRUD.OfferedPrograms[i].getTitle());

Console.SetCursorPosition(30, Y);

Console.WriteLine(DegreeProgramCRUD.OfferedPrograms[i].getDuration());

}

}

public static DegreeProgram TakeInputOfDegreeProgram()

{

string title;

string Duration;

int seats;

Console.WriteLine("Enter Title of Degree ");

title = Console.ReadLine();

Console.WriteLine("Enter Duration of Degree Program = ");

Duration = Console.ReadLine();

Console.WriteLine("Enter Seats in Program ");

seats = int.Parse(Console.ReadLine());

DegreeProgram temp = new DegreeProgram(title, Duration, seats);

return temp;

}

public static void AddSubjectInDegree()

{

int op = DegreeProgramCRUD.OfferedPrograms.Count - 1;// op stores index of DegreeProgram

char TerminationOption = ' ';

while (TerminationOption != 'E' && TerminationOption != 'e')

{

MenuUI.Header();

Console.WriteLine(".....Select Subject.....");

SubjectUI.SubjectHeader();

for (int i = 0; i < SubjectCRUD.AvailableSubjects.Count; i++)

{

SubjectUI.ViewSubject(i);

}

int index = DegreeProgramUI.AddSubjectInWhichDegree();

if (!DegreeProgramCRUD.OfferedPrograms[op].isRedundancyInSubjects(SubjectCRUD.AvailableSubjects[index - 1]))

{

if (!DegreeProgramCRUD.OfferedPrograms[op].isLimitOfSubjectsApproach(SubjectCRUD.AvailableSubjects[index - 1].getCH()))

DegreeProgramCRUD.OfferedPrograms[op].AddSubject(SubjectCRUD.AvailableSubjects[index - 1]);

/\* AddSubjectInDegree(AcedBrach, op);\*///Ch inacedemic branch

else

Console.WriteLine("You Cannot Add More Subjects...");

}

else

Console.WriteLine("......Subject Redundancy....");

Console.WriteLine("To Exit Press E....");

char.TryParse(MenuUI.TakeString(), out TerminationOption);

}

}

}

**MenuUI:**

class MenuUI

{

public static void DisplayMsg(string Msg)

{

Console.WriteLine(Msg);

}

public static string TakeString()

{

return Console.ReadLine();

}

public static void Header()

{

Console.Clear();

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine(" University Admission Management System \*");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

public static string MainMenu()

{

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. Add Subject in System");

Console.WriteLine("9. Exit");

Console.Write("Enter Option: ");

string option;

option = Console.ReadLine();

return option;

}

}

**StudentUI:**

class StudentUI

{

public static void DisplayMsg(string Msg)

{

Console.WriteLine(Msg);

}

public static string TakeString()

{

return Console.ReadLine();

}

public static Student TakeInputOfStudent()

{

Student temp;

string name;

int age;

int fsc;

int ecat;

Console.WriteLine("ENter Your Name ");

name = Console.ReadLine();

Console.WriteLine("Enter Your Age ");

age = int.Parse(Console.ReadLine());

Console.WriteLine("Enter your FSC MArks ");

fsc = int.Parse(Console.ReadLine());

Console.WriteLine("Enter Your ECAT MArks ");

ecat = int.Parse(Console.ReadLine());

temp = new Student(name, age, fsc, ecat);

return temp;

}

public static int WhichSubjetcRegister(Student Source)

{

SubjectHeader();

for (int i = 0; i < Source.getDegreeSubjectCount(); i++)

{

ViewSubject(Source.getSubjectOfDegree(i), i);

}

MenuUI.DisplayMsg("Choose Option...");

int option;

option = int.Parse(MenuUI.TakeString());

return option;

}

public static int WhichStudent()

{

string name;

MenuUI.DisplayMsg("ENter Your NAme...");

name = MenuUI.TakeString();

for (int i = 0; i < StudentCRUD.Candidates.Count; i++)

{

if (StudentCRUD.Candidates[i].getName() == name && StudentCRUD.Candidates[i].isRegistered())

return i;

}

return -1;

}

public static void GenerateMeritList(Student Source)

{

Console.WriteLine(Source.getName() + " is Enrolled in " + Source.getDegreeName());

}

public static void ViewRegisteredStudents()

{

int Y = Console.CursorTop;

Console.SetCursorPosition(0, Y);

Console.Write("Name");

Console.SetCursorPosition(10, Y);

Console.Write("Fsc Marks");

Console.SetCursorPosition(20, Y);

Console.Write("Ecat Marks");

Console.SetCursorPosition(50, Y);

Console.WriteLine("Degree Program");

for (int i = 0; i < StudentCRUD.Candidates.Count; i++)

{

if (StudentCRUD.Candidates[i].isRegistered())

{

Y = Console.CursorTop;

Console.SetCursorPosition(0, Y);

Console.Write(StudentCRUD.Candidates[i].getName());

Console.SetCursorPosition(10, Y);

Console.Write(StudentCRUD.Candidates[i].getFsc().ToString());

Console.SetCursorPosition(20, Y);

Console.Write(StudentCRUD.Candidates[i].getEcat().ToString());

Console.SetCursorPosition(50, Y);

Console.WriteLine(StudentCRUD.Candidates[i].getDegreeName());

}

}

}

public static bool ViewStudentOfSpecificProgram(string DegreeName)

{

bool flag = false;

int Y = Console.CursorTop;

Console.SetCursorPosition(0, Y);

Console.Write("Name");

Console.SetCursorPosition(10, Y);

Console.Write("Fsc Marks");

Console.SetCursorPosition(20, Y);

Console.Write("Ecat Marks");

Console.SetCursorPosition(50, Y);

Console.WriteLine("Degree Program");

for (int i = 0; i < StudentCRUD.Candidates.Count; i++)

{

if (StudentCRUD.Candidates[i].isRegistered())

{

flag = true;

if (StudentCRUD.Candidates[i].getDegreeName() == DegreeName)

{

Y = Console.CursorTop;

Console.SetCursorPosition(0, Y);

Console.Write(StudentCRUD.Candidates[i].getName());

Console.SetCursorPosition(10, Y);

Console.Write(StudentCRUD.Candidates[i].getFsc().ToString());

Console.SetCursorPosition(20, Y);

Console.Write(StudentCRUD.Candidates[i].getEcat().ToString());

Console.SetCursorPosition(50, Y);

Console.WriteLine(StudentCRUD.Candidates[i].getDegreeName());

}

}

}

return flag;

}

public static void ViewFeeHeader()

{

Console.WriteLine("STudent Name\tFee");

}

public static void ViewFee(string name, float fee)

{

Console.WriteLine(name + "\t" + fee);

}

public static List<DegreeProgram> InputOfPreferences() //AcedemicBranch acedBranch)

{

List<DegreeProgram> Preferences = new List<DegreeProgram>();

string terminate = "1";

while (terminate != "E" && terminate != "e")

{

Console.Clear();

Console.WriteLine("Choose Preferences.....");

Console.WriteLine();

Console.WriteLine();

DegreeProgramUI.ViewAllPrograms();

int index = int.Parse(Console.ReadLine());

if (!StudentCRUD.Candidates[StudentCRUD.Candidates.Count - 1].isRedundancyInPreference(Preferences, DegreeProgramCRUD.OfferedPrograms[index - 1]))

Preferences.Add(DegreeProgramCRUD.OfferedPrograms[index - 1]);

else

Console.WriteLine(".....Redundancy Occurs......");

Console.WriteLine("Press E. To Exit");

terminate = Console.ReadLine();

}

return Preferences;

}

public static void ViewSubject(Subject Source, int index)

{

int Y = Console.CursorTop;

Console.SetCursorPosition(0, Y);

Console.Write("{0}. " + Source.getCode(), index + 1);

Console.SetCursorPosition(30, Y);

Console.WriteLine(Source.getCH().ToString());

}

public static void SubjectHeader()

{

int Y = Console.CursorTop;

Console.SetCursorPosition(0, Y);

Console.WriteLine("Code");

Console.SetCursorPosition(30, Y);

Console.WriteLine("CH");

}

}

**SubjectUI:**

class SubjectUI

{

public static void DisplayMsg(string Msg)

{

Console.WriteLine(Msg);

}

public static string TakeString()

{

return Console.ReadLine();

}

public static Subject TakeInputOfSubject()

{

string code;

string SubjectType;

int CH;

int fee;

Console.WriteLine("Enter code of Course ");

code = Console.ReadLine();

Console.WriteLine("ENter SubjetcType ");

SubjectType = Console.ReadLine();

Console.WriteLine("ENter Number of Credit Hours ");

CH = int.Parse(Console.ReadLine());

Console.WriteLine("ENter Fee..");

fee = int.Parse(Console.ReadLine());

Subject temp = new Subject(code, CH, SubjectType, fee);

return temp;

}

public static void ViewSubject(int index)

{

int Y = Console.CursorTop;

Console.SetCursorPosition(0, Y);

Console.Write("{0}. " + SubjectCRUD.AvailableSubjects[index].getCode(), index + 1);

Console.SetCursorPosition(30, Y);

Console.WriteLine(SubjectCRUD.AvailableSubjects[index].getCH().ToString());

}

public static void SubjectHeader()

{

int Y = Console.CursorTop;

Console.SetCursorPosition(0, Y);

Console.WriteLine("Code");

Console.SetCursorPosition(30, Y);

Console.WriteLine("CH");

}

}

**Driver Program:**

class Program

{

static void Option\_1()

{

MenuUI.Header();

if (DegreeProgramCRUD.OfferedPrograms.Count == 0)

{

StudentUI.DisplayMsg(".....First Add a Degree in System.....");

}

else

{

Student temp = StudentUI.TakeInputOfStudent();

StudentCRUD.AddStudentInList(temp);

StudentCRUD.Candidates[StudentCRUD.Candidates.Count - 1].AssignPreferences(StudentUI.InputOfPreferences());

StudentCRUD.WriteToFlle();

}

}

static void Option\_2()

{

MenuUI.Header();

if (SubjectCRUD.AvailableSubjects.Count == 0)

{

SubjectUI.DisplayMsg("....First Add Subjects in System....");

SubjectUI.TakeString();

}

else

{

DegreeProgram temp = DegreeProgramUI.TakeInputOfDegreeProgram();

DegreeProgramCRUD.AddDegreeInList(temp);

DegreeProgramUI.AddSubjectInDegree();

DegreeProgramCRUD.WriteToFlle();

}

}

static void Option\_3()

{

MenuUI.Header();

if (StudentCRUD.Candidates.Count == 0)

{

MenuUI.DisplayMsg(".....No Record Found....");

MenuUI.TakeString();

}

else

{

StudentCRUD.GiveAdmission();

for (int i = 0; i < StudentCRUD.Candidates.Count; i++)

{

if (StudentCRUD.Candidates[i].isRegistered())

{

StudentUI.GenerateMeritList(StudentCRUD.Candidates[i]);

}

}

StudentCRUD.WriteToFlle();

}

MenuUI.TakeString();

}

static void Option\_4()

{

MenuUI.Header();

if (StudentCRUD.Candidates.Count == 0)

{

MenuUI.DisplayMsg(".....No Record Found....");

}

else

StudentUI.ViewRegisteredStudents();

MenuUI.TakeString(); ;

}

static void Option\_5()

{

MenuUI.Header();

MenuUI.DisplayMsg("Enter Degree Name");

string name = MenuUI.TakeString();

if (!StudentUI.ViewStudentOfSpecificProgram(name))

MenuUI.DisplayMsg(".....No Record FOund....");

}

static void Option\_6()

{

MenuUI.Header();

if (StudentCRUD.Candidates.Count == 0)

MenuUI.DisplayMsg("......No Student Found.......");

else if (SubjectCRUD.AvailableSubjects.Count == 0)

MenuUI.DisplayMsg(".....No Subject is Available.....");

else

{

int stu = StudentUI.WhichStudent();

if (stu != -1)

{

int index1 = StudentUI.WhichSubjetcRegister(StudentCRUD.Candidates[stu]);

if (StudentCRUD.Candidates[index1 - 1].RegisterSubject(SubjectCRUD.AvailableSubjects[index1 - 1]))

{

MenuUI.DisplayMsg("Subject Has been Registered...");

StudentCRUD.WriteToFlle();

}

else

MenuUI.DisplayMsg("You cannot register More Subjetcs...");

}

else

MenuUI.DisplayMsg("Student Is not registered");

}

}

static void Option\_7()

{

MenuUI.Header();

StudentUI.ViewFeeHeader();

for (int i = 0; i < StudentCRUD.Candidates.Count; i++)

{

StudentUI.ViewFee(StudentCRUD.getStudent(i).getName(), StudentCRUD.getStudent(i).CalculateDues());

}

}

static void Option\_8()

{

MenuUI.Header();

SubjectCRUD.AddSubjectInList(SubjectUI.TakeInputOfSubject());

SubjectCRUD.WriteToFlle();

}

static void Main(string[] args)

{

SubjectCRUD.ReadFromFile();

DegreeProgramCRUD.ReadFromFile();

StudentCRUD.ReadFromFile();

string option;

do

{

MenuUI.Header();

option = MenuUI.MainMenu();

switch (option)

{

case "1":

Option\_1();

break;

case "2":

Option\_2();

break;

case "3":

Option\_3();

break;

case "4":

Option\_4();

break;

case "5":

Option\_5();

break;

case "6":

Option\_6();

break;

case "7":

Option\_7();

break;

case "8":

Option\_8();

break;

}

MenuUI.TakeString();

} while (option != "9");

Console.Read();

}

}

**Sequence Diagram**