**Assignment: Multiple Classes with Association**



Session: 2021 – 2024

**Submitted by:**

Bisma Muhammad Ali 2021-CS-170

**Supervised by:**

Madam Maida

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

**Table Of Contents**

1. Point Of Sale Application …………………………Pg No.3
2. Ocean Navigation………………………………… Pg No.19
3. UAMS……………………………………………. Pg No. 31
4. CofeeShop…………………………………………Pg No. 51
5. Point And Line………………………………….…Pg No. 65
6. GameObject………………………………………. Pg No.77

**Scenerio1: Point of Sale Application**

**Case Study Scenario:**

Miss Client wants to develop a software system for her departmental store. She wants this

system to have the following functionalities.

As an Admin, she can

Add Products.

● View All Products.

● Find Product with Highest Unit Price.

● View Sales Tax of All Products.

● Products to be Ordered. (less than threshold)

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

Name of Product. Product Category. Product Price. Available Stock Quantity. Minimum

Stock threshold Quantity after which the owner wants to order the product.

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

if there is any other type the tax is 15%

She also wants that

1. The Customers to view all the products

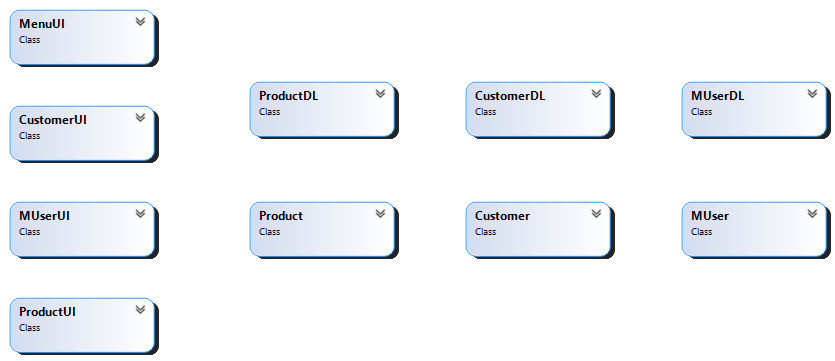
2. Customers can buy the products (When a customer buy a product then its

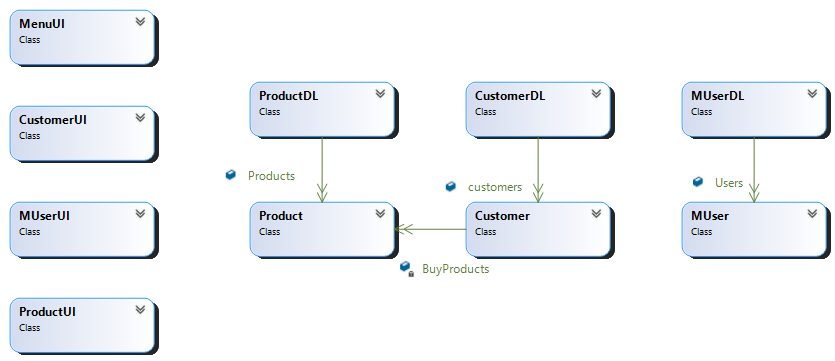
quantity should decrease from the stock)

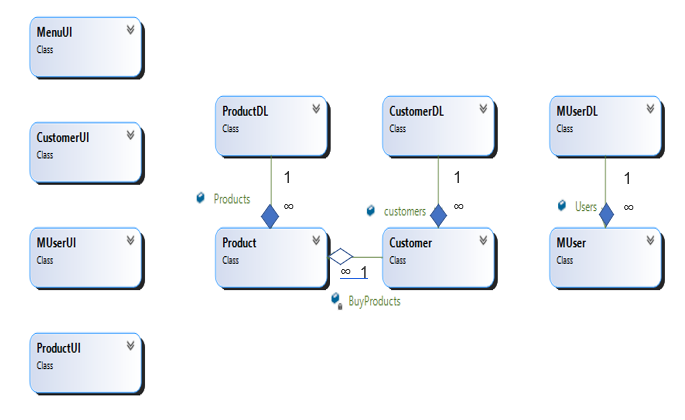
3. Generate invoice (While calculating the price of the products that the

customer has bought, sales tax should be applied.)

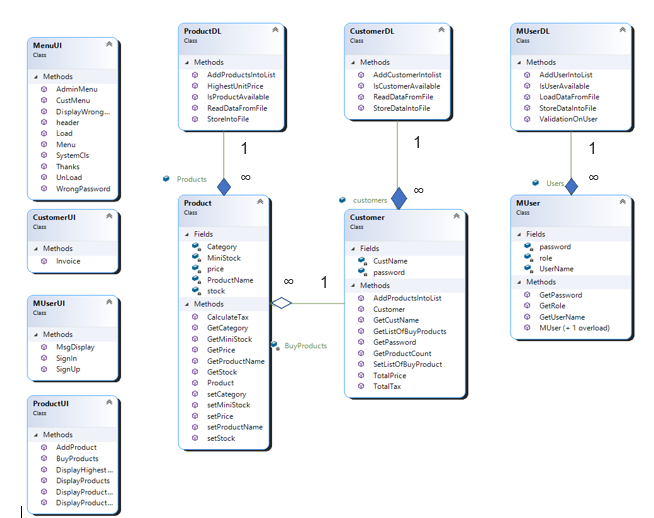
**Domain Model:**

1. ****Domain Model with classes name.
2. Domain Model with relations and Constraints.



3. Domain Model with multiplicity

**Class Diagram:**



**Full Code**

**BL Classes:**

1. **Customer**

public Customer(string CustName, string password)

{

this.CustName = CustName;

this.password = password;

BuyProducts = new List<Product>();

}

private string CustName;

private string password;

private List<Product> BuyProducts;

public void AddProductsIntoList(Product p)

{

BuyProducts.Add(p);

}

public double TotalTax()

{

double TotTax=0;

foreach(Product p in BuyProducts)

{

TotTax = TotTax + p.CalculateTax();

}

return TotTax;

}

public double TotalPrice()

{

double TotPrice = 0;

foreach (Product p in BuyProducts)

{

TotPrice = TotPrice + p.GetPrice();

}

return TotPrice;

}

public string GetCustName()

{

return CustName;

}

public string GetPassword()

{

return password;

}

public List<Product> GetListOfBuyProducts()

{

return BuyProducts;

}

public void SetListOfBuyProduct(List<Product>BuyProducts)

{

this.BuyProducts = BuyProducts;

}

public int GetProductCount()

{

return BuyProducts.Count;

}

}

}

1. **Muser**

public MUser(string UserName, string password, string role)

{

this.UserName = UserName;

this.password = password;

this.role = role;

}

public MUser(string UserName, string password)

{

this.UserName = UserName;

this.password = password;

this.role = null;

}

private string UserName;

private string password;

private string role;

public string GetUserName()

{

return UserName;

}

public string GetPassword()

{

return password;

}

public string GetRole()

{

return role;

}

}

1. **Product:**

public Product(string ProductName, string Category, float price, int stock, int MiniStock)

{

this.ProductName = ProductName;

this.Category = Category;

this.price = price;

this.stock = stock;

this.MiniStock = MiniStock;

}

private string ProductName;

private string Category;

private float price;

private int stock;

private int MiniStock;

public double CalculateTax()

{

double tax;

if(Category == "FRUIT" || Category == "fruite" || Category == "Fruit")

{

tax = 10F;

}

else if(Category == "Grocery" || Category == "grocery" || Category == "Grocery")

{

tax = 5F;

}

else

{

tax = 15F;

}

tax = price \* tax / 100;

return tax;

}

public string GetProductName()

{

return ProductName;

}

public string GetCategory()

{

return Category;

}

public float GetPrice()

{

return price;

}

public int GetStock()

{

return stock;

}

public int GetMiniStock()

{

return MiniStock;

}

public void setProductName(string ProductName)

{

this.ProductName = ProductName;

}

public void setCategory(string Category)

{

this.Category = Category;

}

public void setPrice(float price)

{

this.price = price;

}

public void setStock(int stock)

{

this.stock = stock;

}

public void setMiniStock(int MiniStock)

{

this.MiniStock = MiniStock;

}

}

}

**DL Classes:**

1. **CustomerDL:**

class CustomerDL

{

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

public static void AddCustomerIntolist(Customer c)

{

customers.Add(c);

}

public static Customer IsCustomerAvailable(MUser cust)

{

foreach (Customer c in customers)

{

if (cust.GetUserName() == c.GetCustName() && cust.GetPassword() == c.GetPassword())

{

return c;

}

}

return null;

}

public static void StoreDataIntoFile()

{

string path = "Customers.txt";

StreamWriter file = new StreamWriter(path, false);

foreach (Customer cust in customers)

{

string item = "";

if (cust.GetProductCount() > 0)

{

List<Product> products = cust.GetListOfBuyProducts();

for (int x = 0; x < cust.GetProductCount() - 1; x++)

{

item = item + products[x].GetProductName() + ",";

}

item = item + products[cust.GetProductCount() - 1].GetProductName();

file.WriteLine(cust.GetCustName() + "," + cust.GetPassword() + ","+ item);

}

}

file.Flush();

file.Close();

}

public static bool ReadDataFromFile()

{

string path = "Customers.txt";

if (File.Exists(path))

{

StreamReader file = new StreamReader(path);

string record;

while ((record = file.ReadLine()) != null)

{

string[] splitRecord = record.Split(',');

string CustName = splitRecord[0];

string password = splitRecord[1];

MUser user = new MUser(CustName, password);

Customer cust = IsCustomerAvailable(user);

if (cust != null)

{

for (int i = 2; i < splitRecord.Length; i++)

{

Product p = ProductDL.IsProductAvailable(splitRecord[i]);

if (p != null)

{

cust.AddProductsIntoList(p);

}

}

}

}

file.Close();

return true;

}

return false;

}

}

}

1. **MUserDL:**

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

public static void AddUserIntoList(MUser user)

{

Users.Add(user);

}

public static MUser IsUserAvailable(MUser user)

{

foreach(MUser u in Users)

{

if(user.GetUserName()== u.GetUserName() && user.GetPassword() == u.GetPassword())

{

return u;

}

}

return null;

}

public static bool LoadDataFromFile()

{

string path = "Users.txt";

if (File.Exists(path))

{

StreamReader user = new StreamReader(path);

string record;

while ((record = user.ReadLine()) != null)

{

string[] splitRecord = record.Split(',');

string UserName = splitRecord[0];

string password = splitRecord[1];

string role = splitRecord[2];

MUser u = new MUser(UserName, password, role);

AddUserIntoList(u);

if(role == "Customer" || role == "customer")

{

Customer cust = new Customer(UserName, password);

CustomerDL.AddCustomerIntolist(cust);

}

}

user.Close();

return true;

}

return false;

}

public static void StoreDataIntoFile(MUser user)

{

string path = "Users.txt";

StreamWriter u= new StreamWriter(path, true);

u.Write(user.GetUserName() + "," + user.GetPassword() + "," + user.GetRole());

u.Flush();

u.Close();

}

public static bool ValidationOnUser(MUser user)

{

foreach (MUser u in Users)

{

if (user.GetUserName() == u.GetUserName() && user.GetPassword() == u.GetPassword())

{

return false;

}

}

return true;

}

}

}

1. **ProductDL:**

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

public static void AddProductsIntoList(Product p)

{

Products.Add(p);

}

public static Product IsProductAvailable(string name)

{

foreach(Product p in Products)

{

if(name== p.GetProductName())

{

return p;

}

}

return null;

}

public static Product HighestUnitPrice()

{

List<Product> SortedList = Products.OrderByDescending(o => o.GetPrice()).ToList();

return SortedList[0];

}

public static void StoreIntoFile()

{

string path = "Products.txt";

StreamWriter file = new StreamWriter(path, false);

foreach (Product p in Products)

{

file.WriteLine(p.GetProductName() + "," + p.GetCategory() + "," + p.GetPrice() + "," + p.GetStock() + "," + p.GetMiniStock());

}

file.Flush();

file.Close();

}

public static bool ReadDataFromFile()

{

string path = "Products.txt";

if (File.Exists(path))

{

StreamReader file = new StreamReader(path);

string record;

while ((record = file.ReadLine()) != null)

{

string[] splitRecord = record.Split(',');

string ProductName = splitRecord[0];

string Category = splitRecord[1];

float price = float.Parse(splitRecord[2]);

int stock = int.Parse(splitRecord[3]);

int MiniStock = int.Parse(splitRecord[4]);

Product p = new Product(ProductName, Category, price, stock, MiniStock);

AddProductsIntoList(p);

}

file.Close();

return true;

}

return false;

}

**UI Classes:**

1. **CustomerUI:**

public static void Invoice(Customer cust)

{

Console.WriteLine("Customer name: " + cust.GetCustName());

double price = cust.TotalPrice() + cust.TotalTax();

Console.WriteLine("Total Price After applying tax: " + price);

}

1. **MUserUI:**

public static MUser SignUp()

{

Console.WriteLine("Enter Your name: ");

string UserName = Console.ReadLine();

Console.WriteLine("Enter Your Password: ");

string password = Console.ReadLine();

Console.WriteLine("Enter your role: ");

string role = Console.ReadLine();

MUser user = new MUser(UserName, password, role);

if (MUserDL.ValidationOnUser(user))

{

return user;

}

return null;

}

public static void MsgDisplay()

{

Console.WriteLine("This username and Password are already in used");

}

public static MUser SignIn()

{

Console.WriteLine("Enter Your name: ");

string UserName = Console.ReadLine();

Console.WriteLine("Enter Your Password: ");

string password = Console.ReadLine();

MUser user = new MUser(UserName, password);

return MUserDL.IsUserAvailable(user);

}

}

}

1. **ProductUI:**

public static Product AddProduct()

{

Console.WriteLine("Enter the name of product: ");

string ProductName = Console.ReadLine();

Console.WriteLine("Enter the category of product: ");

string Category = Console.ReadLine();

Console.WriteLine("Enter the price of product: ");

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

Console.WriteLine("Enter the Available stock of product: ");

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

Console.WriteLine("Enter the Threshold stock of product: ");

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

Product p = new Product(ProductName, Category, price, stock, MiniStock);

return p;

}

public static void DisplayProducts()

{

foreach (Product p in ProductDL.Products)

{

Console.WriteLine(" Item name: " + p.GetProductName());

Console.WriteLine(" Item Type: " + p.GetCategory());

Console.WriteLine(" item Price: " + p.GetPrice());

Console.WriteLine(" item stock: " + p.GetStock());

Console.WriteLine("\n");

}

}

public static void DisplayProductsWithTax()

{

foreach (Product p in ProductDL.Products)

{

Console.WriteLine(" Item name: " + p.GetProductName());

Console.WriteLine(" Item Type: " + p.GetCategory());

Console.WriteLine(" item Price: " + p.GetPrice());

Console.WriteLine(" item stock: " + p.GetStock());

Console.WriteLine(" item tax " + p.CalculateTax());

Console.WriteLine("\n");

}

}

public static void DisplayHighestPricePtoduct()

{

Product p = ProductDL.HighestUnitPrice();

Console.WriteLine(" Item name: " + p.GetProductName());

Console.WriteLine(" Item Type: " + p.GetCategory());

Console.WriteLine(" item Price: " + p.GetPrice());

Console.WriteLine(" item stock: " + p.GetStock());

Console.WriteLine("\n");

}

public static void DisplayProductsLessthanMiniStock()

{

foreach (Product p in ProductDL.Products)

{

if (p.GetStock() < p.GetMiniStock())

{

Console.WriteLine(" Item name: " + p.GetProductName());

Console.WriteLine(" Item Type: " + p.GetCategory());

Console.WriteLine(" item Price: " + p.GetPrice());

Console.WriteLine(" item stock: " + p.GetStock());

Console.WriteLine("\n");

}

}

}

public static void BuyProducts(Customer cust)

{

int stock;

Console.WriteLine("Enter the name of product you want to buy: ");

string ProductName = Console.ReadLine();

Product product = ProductDL.IsProductAvailable(ProductName);

if (product != null)

{

cust.AddProductsIntoList(product);

stock=product.GetStock();

product.setStock(stock-1);

}

}

1. **MenuUI :**

public static void header()

{

Console.WriteLine("////////////////////////////////////////////////////////////");

Console.WriteLine("-- POINT AND SALE APPLICATION --");

Console.WriteLine("////////////////////////////////////////////////////////////");

Console.WriteLine(" ");

}

// end of 2nd header function

public static void SystemCls()

{

Console.WriteLine("Press any key to continue....");

Console.ReadKey();

Console.Clear();

}

public static int CustMenu()

{

header();

Console.WriteLine("//////// Main Menu //////////// ");

Console.WriteLine("Select one of the following options number . . .");

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

Console.WriteLine("2. Buy new Products");

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

Console.WriteLine("4. Exit");

Console.Write("Choose your option: ");

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

return option;

}

public static char Menu()

{

header();

Console.WriteLine("Press 1 to Sign-In");

Console.WriteLine("Press 2 to Sign-Up");

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

char op;

op = char.Parse(Console.ReadLine());

SystemCls();

return op;

}

public static void WrongPassword()

{

Console.WriteLine("You Entered Wrong Password");

}

static public int AdminMenu()

{

header();

Console.WriteLine("//////// Main Menu //////////// ");

Console.WriteLine("Select one of the following options number . . .");

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

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

Console.WriteLine("3. Find Product with Highest unit price");

Console.WriteLine("4. View sales tax of All Products");

Console.WriteLine("5. View Products to be orderd");

Console.WriteLine("6. Exit");

Console.Write("Choose your option: ");

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

return option;

}

public static void DisplayWrongOption()

{

Console.WriteLine("You choose wrong option");

}

public static void Thanks()

{

Console.WriteLine("Thank you for using our application");

}

public static void Load()

{

Console.WriteLine("File loaded Successfully");

Console.ReadKey();

}

public static void UnLoad()

{

Console.WriteLine("File does not loaded Successfully");

Console.ReadKey();

}

}

}

**Driver Program:**

static void Main(string[] args)

{

if( MUserDL.LoadDataFromFile())

{

MenuUI.Load();

}

else

{

MenuUI.UnLoad();

}

if (ProductDL.ReadDataFromFile())

{

MenuUI.Load();

}

else

{

MenuUI.UnLoad();

}

if (CustomerDL.ReadDataFromFile())

{

MenuUI.Load();

}

else

{

MenuUI.UnLoad();

}

char option;

while (true)

{

option = MenuUI.Menu();

if (option == '1')

{

MUser user = MUserUI.SignIn();

MenuUI.SystemCls();

if (user != null)

{

if (user.GetRole() == "Admin" || user.GetRole() == "admin")

{

while (true)

{

int opt = MenuUI.AdminMenu();

MenuUI.SystemCls();

MenuUI.header();

if (opt == 1)

{

Product p = ProductUI.AddProduct();

ProductDL.AddProductsIntoList(p);

ProductDL.StoreIntoFile();

}

else if (opt == 2)

{

ProductUI.DisplayProducts();

}

else if (opt == 3)

{

ProductUI.DisplayHighestPricePtoduct();

}

else if (opt == 4)

{

ProductUI.DisplayProductsWithTax();

}

else if (opt == 5)

{

ProductUI.DisplayProductsLessthanMiniStock();

}

else if (opt == 6)

{

break;

}

else

{

MenuUI.DisplayWrongOption();

}

MenuUI.SystemCls();

}

}

else if (user.GetRole() == "Customer" || user.GetRole() == "customer")

{

Customer cust = CustomerDL.IsCustomerAvailable(user);

while (true)

{

int opt = MenuUI.CustMenu();

MenuUI.SystemCls();

MenuUI.header();

if (opt == 1)

{

ProductUI.DisplayProducts();

}

else if (opt == 2)

{

ProductUI.BuyProducts(cust);

ProductDL.StoreIntoFile();

CustomerDL.StoreDataIntoFile();

}

else if (opt == 3)

{

CustomerUI.Invoice(cust);

}

else if (opt == 4)

{

break;

}

else

{

MenuUI.DisplayWrongOption();

}

MenuUI.SystemCls();

}

}

}

else

MenuUI.WrongPassword();

}

else if(option == '2')

{

MenuUI.header();

MUser user = MUserUI.SignUp();

if (user != null)

{

MUserDL.AddUserIntoList(user);

MUserDL.StoreDataIntoFile(user);

if (user.GetRole() == "Customer" || user.GetRole() == "customer")

{

Customer cust = new Customer(user.GetUserName(), user.GetPassword());

CustomerDL.AddCustomerIntolist(cust);

}

}

else

MUserUI.MsgDisplay();

}

else if(option == '3')

{

MenuUI.Thanks();

MenuUI.SystemCls();

break;

}

else

{

MenuUI.DisplayWrongOption();

}

MenuUI.SystemCls();

}

}

}

}

**Scenerio2: Ocean Navigation**

**Case Study Scenario:**

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.

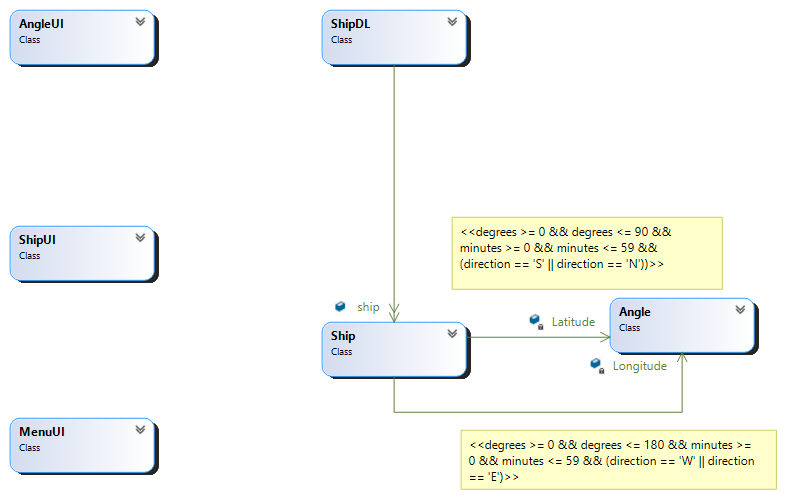
Write one member function to change the angle value (in degrees and minutes) and a direction given from the user, and a second to display the angle value in 179°59.9’ E in string format. Also write a three-argument constructor. You can use this to print a degree (°) symbol. Console.WriteLine("\u00b0");

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.

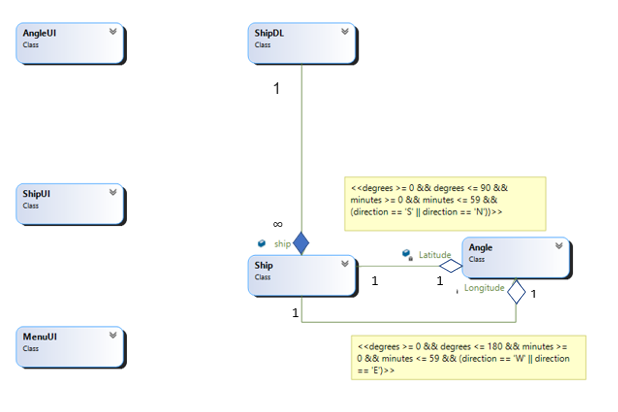
**Domain Model:**

1. Domain Model with classes name

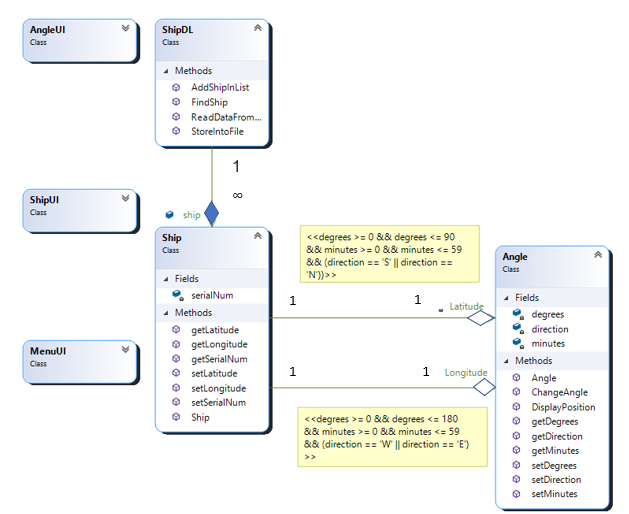
****

1. Domain Model with relations and Constraints.

3. Domain Model with multiplicity

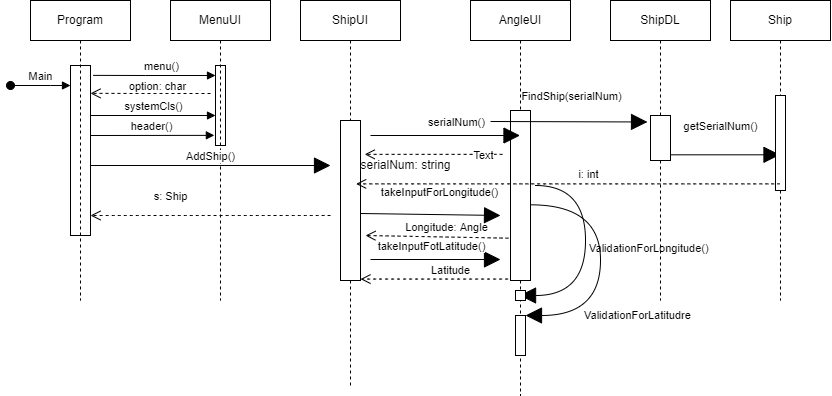


**Class Diagram:**

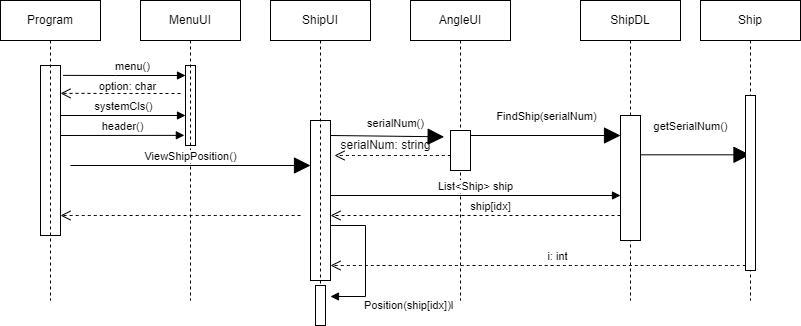


**Sequence Diagrams:**

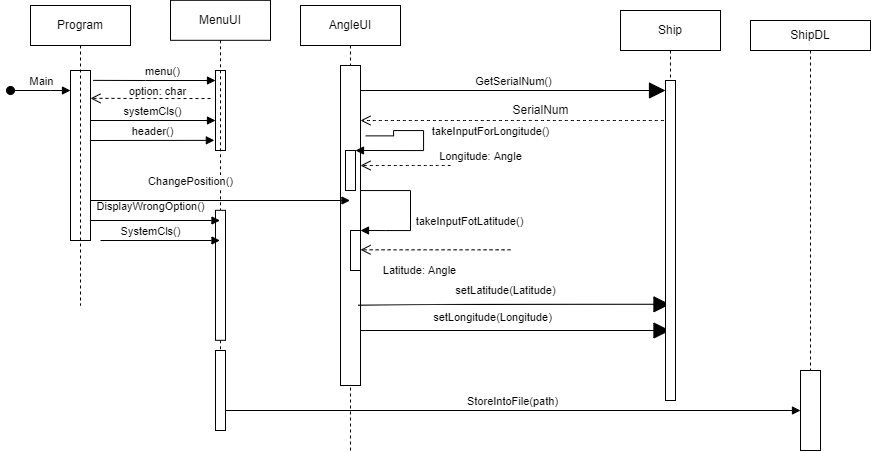
1. Add Ship



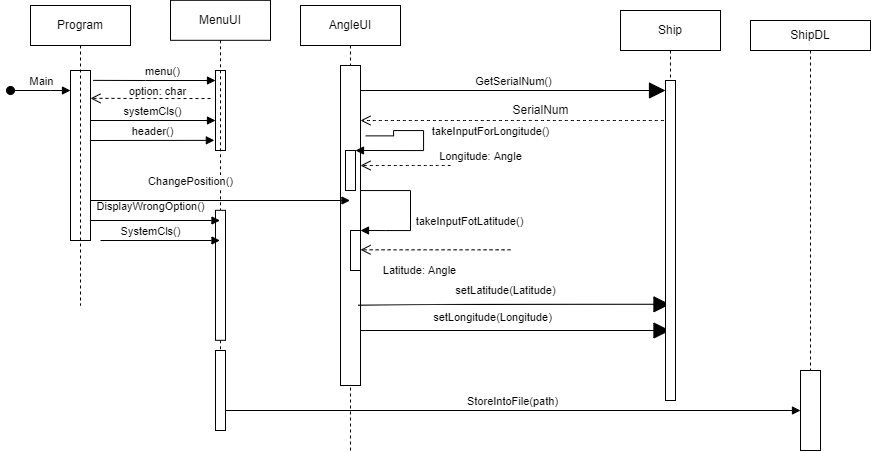
1. View Ship Position



1. Get serial number



1. Change Position



**Full Code**

**BL Classes:**

1. **Angle**

public Angle(int degrees, float minutes, char direction)

{

this.degrees = degrees;

this.minutes = minutes;

this.direction = direction;

}

private int degrees;

private float minutes;

private char direction;

public int getDegrees()

{

return degrees;

}

public float getMinutes()

{

return minutes;

}

public char getDirection()

{

return direction;

}

public void setDegrees(int degrees)

{

this.degrees = degrees;

}

public void setMinutes(float minutes)

{

this.minutes = minutes;

}

public void setDirection(char direction)

{

this.direction = direction;

}

public void ChangeAngle(int degrees, float minutes, char direction)

{

this.degrees = degrees;

this.minutes = minutes;

this.direction = direction;

}

public string DisplayPosition()

{

return ( degrees + "\u00b0" + minutes + "\'" + direction);

}

}

}

1. **Ship:**

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

{

this.serialNum = serialNum;

this.Longitude = Longitude;

this.Latitude = Latitude;

}

private string serialNum;

private Angle Longitude;

private Angle Latitude;

public string getSerialNum()

{

return serialNum;

}

public Angle getLongitude()

{

return Longitude;

}

public Angle getLatitude()

{

return Latitude;

}

public void setSerialNum(string serialNum)

{

this.serialNum = serialNum;

}

public void setLongitude(Angle Longitude)

{

this.Longitude = Longitude;

}

public void setLatitude(Angle Latitude)

{

this.Latitude = Latitude;

}

}

**DL Classes:**

1. **ShipDL:**

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

public static void AddShipInList(Ship s)

{

ship.Add(s);

}

public static int FindShip(string serialNum)

{

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

{

if (serialNum == ship[i].getSerialNum())

{

return i;

}

}

return -1;

}

public static void StoreIntoFile(string path)

{

StreamWriter file = new StreamWriter(path, false);

foreach (Ship s in ship)

{

string Longitude, Latitude;

Latitude = s.getLatitude().getDegrees() + "," + s.getLatitude().getMinutes() + "," + s.getLatitude().getDirection();

Longitude = s.getLongitude().getDegrees() + "," + s.getLongitude().getMinutes() + "," + s.getLongitude().getDirection();

file.WriteLine(s.getSerialNum() + "," + Latitude + "," + Longitude);

}

file.Flush();

file.Close();

}

public static bool ReadDataFromFile(string path)

{

StreamReader file = new StreamReader(path);

if (File.Exists(path))

{

string record;

while ((record = file.ReadLine()) != null)

{

string[] splittedRecord = record.Split(',');

string serialNumber = splittedRecord[0];

int LatDegree = int.Parse(splittedRecord[1]);

float LatMinutes = float.Parse(splittedRecord[2]);

char LatDirection = char.Parse(splittedRecord[3]);

int LonDegree = int.Parse(splittedRecord[4]);

float LonMinutes = float.Parse(splittedRecord[5]);

char LonDirection = char.Parse(splittedRecord[6]);

Angle Longitude = new Angle(LonDegree, LonMinutes, LonDirection);

Angle Latitude = new Angle(LatDegree, LatMinutes, LatDirection);

Ship s = new Ship(serialNumber, Longitude, Latitude);

AddShipInList(s);

}

file.Close();

return true;

}

return false;

}

**UI Classes:**

1. **AngleUI:**

public static Angle takeInputforLongitude()

{

while (true)

{

Console.WriteLine(" ");

Console.WriteLine("Enter Positions for Longitude: \n");

Console.Write("Enter Position in degrees: ");

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

Console.Write("Enter Position in minutes: ");

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

Console.Write("Enter Direction of ship: ");

char direction = char.Parse(Console.ReadLine());

if (ValidationforLongitude(degrees, minutes, direction))

{

Angle Longitude = new Angle(degrees, minutes, direction);

return Longitude;

}

else

{

Console.WriteLine("please enter valid position");

MenuUI.systemCls();

}

}

}

public static Angle takeInputforLatitude()

{

while (true)

{

Console.WriteLine(" ");

Console.WriteLine("Enter Positions for Latitude: \n");

Console.Write("Enter Position in degrees: ");

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

Console.Write("Enter Position in minutes: ");

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

Console.Write("Enter Direction of ship: ");

char direction = char.Parse(Console.ReadLine());

if (ValidationforLatitude(degrees, minutes, direction))

{

Angle Latitude = new Angle(degrees, minutes, direction);

return Latitude;

}

else

{

Console.WriteLine("please enter valid position");

}

}

}

public static string serialNum()

{

Console.Write("Enter ship's serial number: ");

string serialNum = Console.ReadLine();

return serialNum;

}

public static bool ValidationforLongitude(int degrees, float minutes, char direction)

{

if (degrees >= 0 && degrees <= 180 && minutes >= 0 && minutes <= 59 && (direction == 'W' || direction == 'E'))

{

return true;

}

return false;

}

public static bool ValidationforLatitude(int degrees, float minutes, char direction)

{

if (degrees >= 0 && degrees <= 90 && minutes >= 0 && minutes <= 59 && (direction == 'S' || direction == 'N'))

{

return true;

}

return false;

}

public static void ChangePosition()

{

int count = 0;

string serNo = serialNum();

foreach (Ship s in ShipDL.ship)

{

if (s.getSerialNum() == serNo)

{

count++;

s.setLatitude(takeInputforLatitude());

s.setLongitude(takeInputforLongitude());

Console.WriteLine("Position is changed");

break;

}

}

if (count == 0)

{

Console.WriteLine("no ship available of this serial number");

}

}

1. **ShipUI:**

string serialNum;

while (true)

{

serialNum = AngleUI.serialNum();

if (ShipDL.FindShip(serialNum) == -1)

{

break;

}

else

{

Console.WriteLine("You already assign this serial number to another ship");

MenuUI.systemCls();

}

}

Angle Longitude = AngleUI.takeInputforLongitude();

Angle Latitude = AngleUI.takeInputforLatitude();

Ship s = new Ship(serialNum, Longitude, Latitude);

return s;

}

public static void GetSerialNum()

{

Angle Longitude = AngleUI.takeInputforLongitude();

Angle Latitude = AngleUI.takeInputforLatitude();

bool IsAvailable = false;

foreach (Ship p in ShipDL.ship)

{

if (p.getLongitude().DisplayPosition() == Longitude.DisplayPosition() && p.getLatitude().DisplayPosition() == Latitude.DisplayPosition())

{

diplaySerialNum(p);

IsAvailable = true;

break;

}

}

if (IsAvailable != true)

{

Console.WriteLine("no ship Available at this position");

}

}

public static void Position(Ship s)

{

Console.WriteLine("Longitud of a ship: " + s.getLongitude().DisplayPosition());

Console.WriteLine("Latitude of a ship: " + s.getLatitude().DisplayPosition());

}

public static void diplaySerialNum(Ship s)

{

Console.WriteLine("serial number of ship: " + s.getSerialNum());

}

public static void ViewShipPosition()

{

string serialNum = AngleUI.serialNum();

int idx = ShipDL.FindShip(serialNum);

if (idx != -1)

Position(ShipDL.ship[idx]);

else

Console.WriteLine("no ship Available of this serial number");

}

1. **MenuUI :**

public static void header()

{

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

Console.WriteLine("\* OCEAN NAVIGATION MANAGEMENT SYSTEM \*");

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

Console.WriteLine(" ");

}

public static char menu()

{

char option;

header();

Console.WriteLine("////////// MAIN MENU /////////////// ");

Console.WriteLine(" ");

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

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

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

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

Console.WriteLine("Press 5 to EXIT ");

Console.WriteLine("Enter your option ");

return option = char.Parse(Console.ReadLine());

public static void systemCls()

{

Console.WriteLine("Press any key to continue....");

Console.ReadKey();

Console.Clear();

}

public static void WrongOption()

{

Console.WriteLine("You choose wrong option");

}

public static void Load()

{

Console.WriteLine("File loaded Successfully");

Console.ReadKey();

Console.Clear();

}

public static void UnLoad()

{

Console.WriteLine("File does not loaded Successfully");

Console.ReadKey();

Console.Clear();

}

**Driver Program:**

string path = "ship.txt";

if (ShipDL.ReadDataFromFile(path))

{

MenuUI.Load();

}

else

{

MenuUI.UnLoad();

}

while (true)

{

char option = MenuUI.menu();

MenuUI.systemCls();

MenuUI.header();

if (option == '1')

{

Ship s = ShipUI.AddShip();

ShipDL.AddShipInList(s);

ShipDL.StoreIntoFile(path);

}

else if (option == '2')

{

ShipUI.ViewShipPosition();

}

else if (option == '3')

{

ShipUI.GetSerialNum();

}

else if (option == '4')

{

AngleUI.ChangePosition();

ShipDL.StoreIntoFile(path);

}

else if (option == '5')

{

break;

}

else

{

MenuUI.WrongOption();

}

MenuUI.systemCls();

}

**Scenerio1: UAMS**

**Case Study Scenario:**

The academic branch offers different programs within different departments each

program has a degree title and duration of the degree.

Student Apply for admission to University and provides /her name, age, FSC, and

Ecat Marks and selects any number of preferences among the available programs.

The admission department prepares a merit list according to the highest merit and

available seats and registers selected students in the program.

The academic Branch also adds subjects for each program. A subject has subject code,

credit hours, subject type, and subjectFee A Program cannot have more than 20

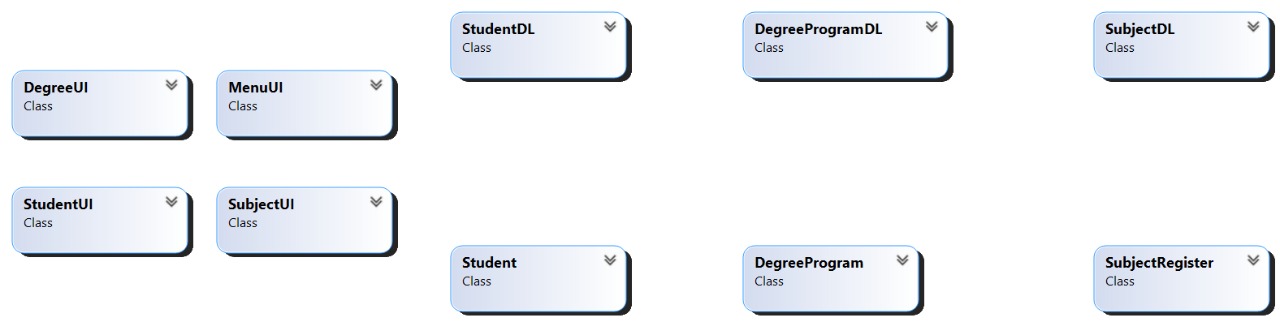
Credit hour subjects. A Student Registers for multiple subjects but only from his

enrolled program’s subject but he/she can not take more than 9 credit hours.

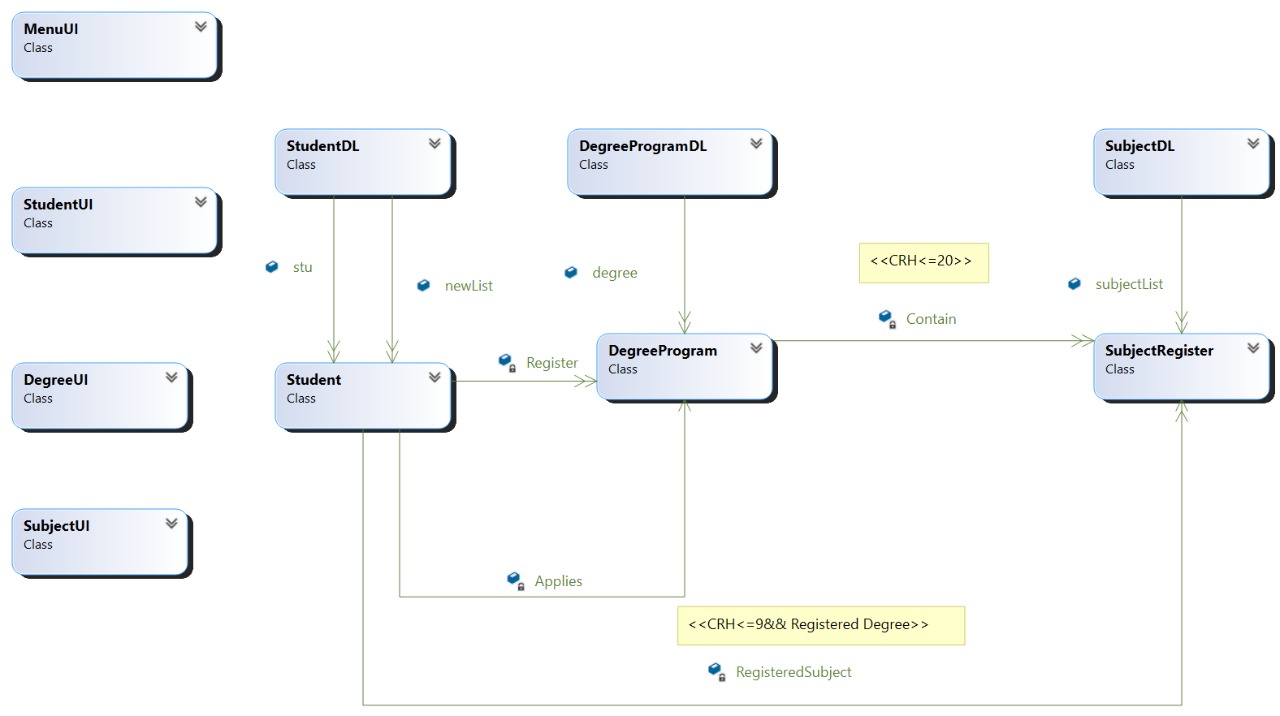
The fee department generates fees according to the registered subjects of the students.

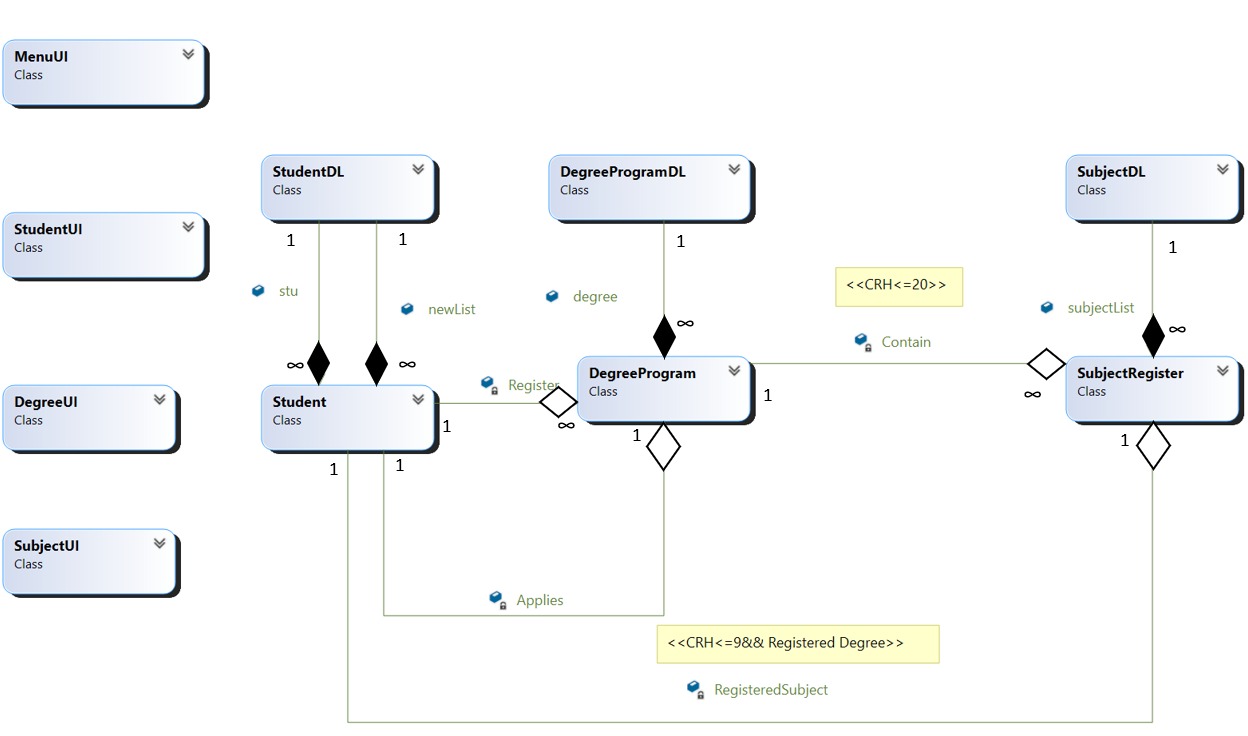
**Domain Model:**

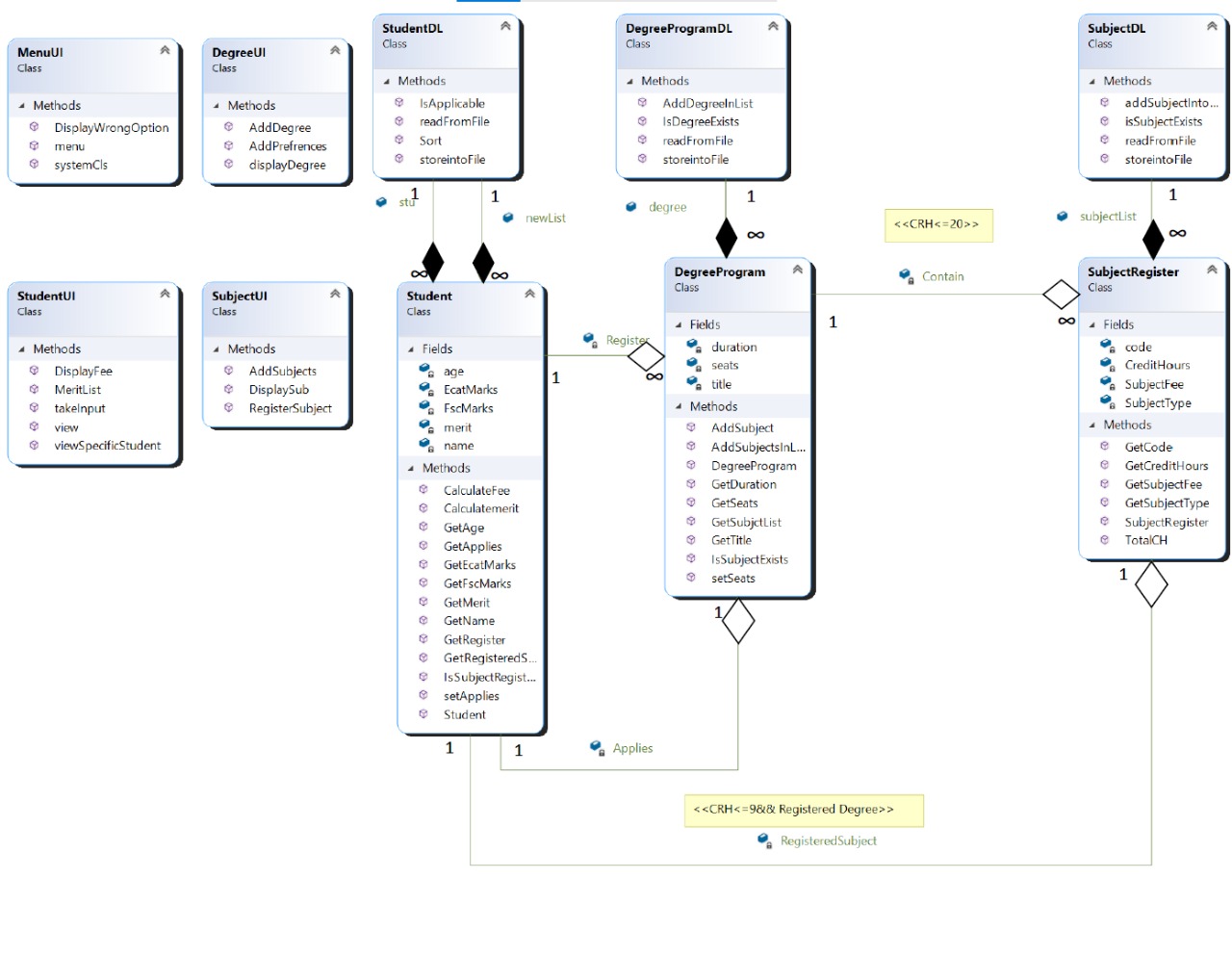
1. Domain Model with classes name.

****

1. Domain Model with relations and Constraints

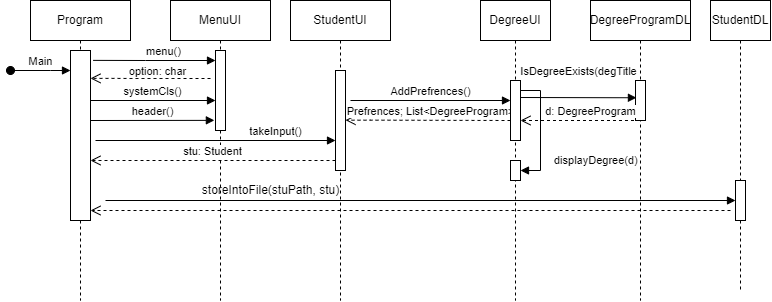
.

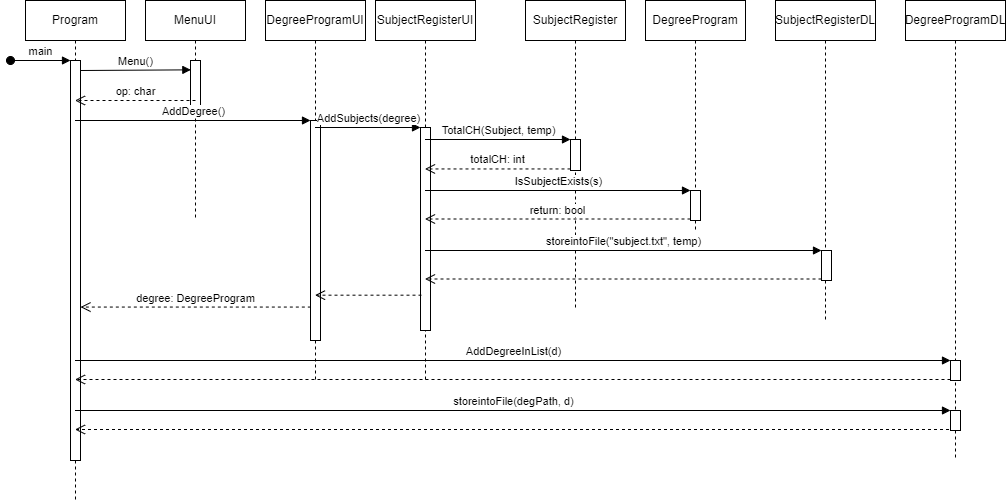
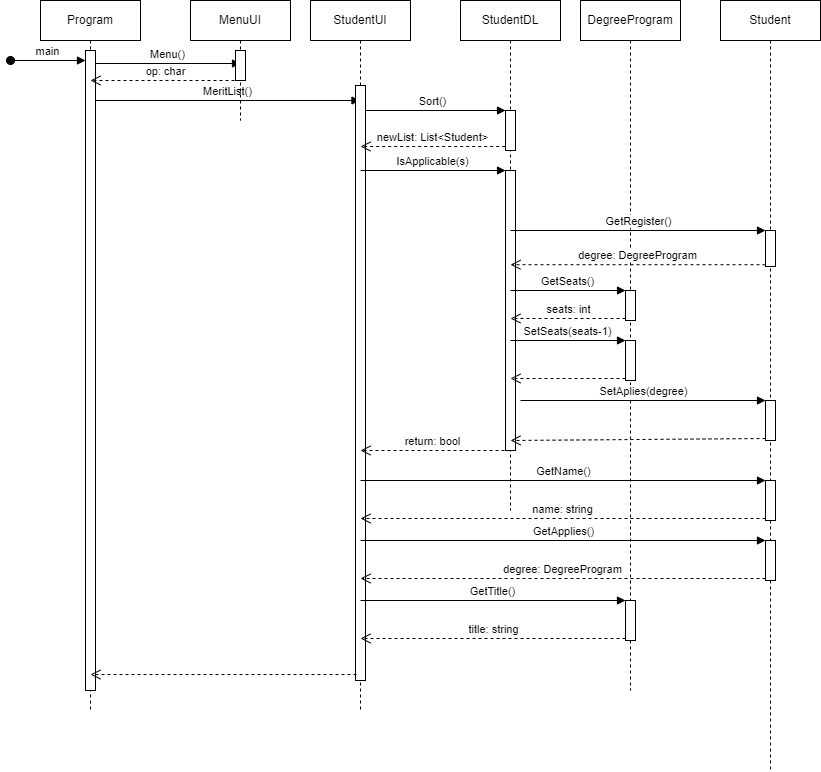
3. Domain Model with multiplicity

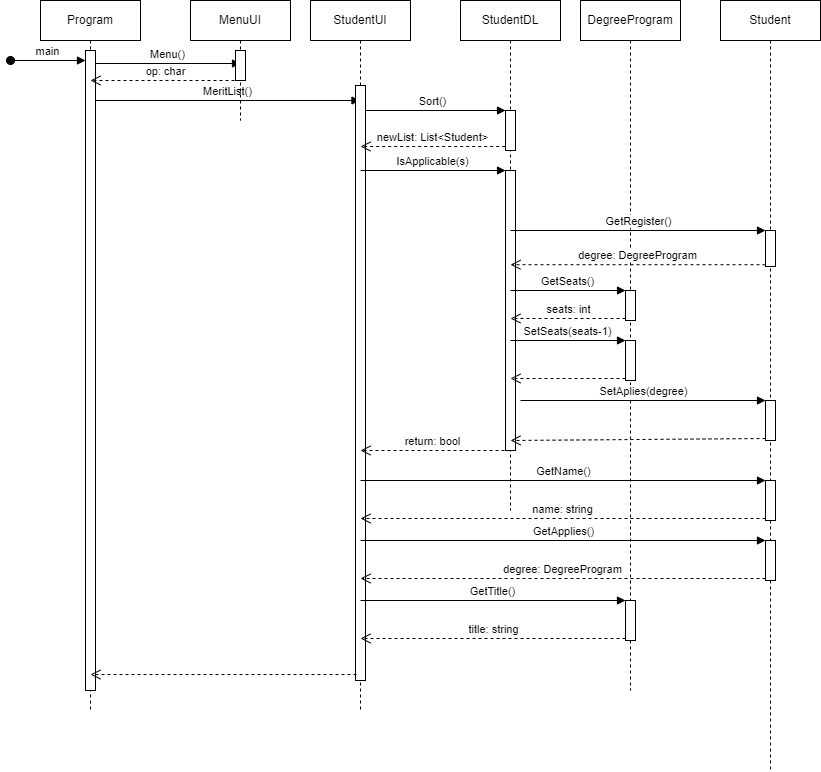
**ClassDiagram:**

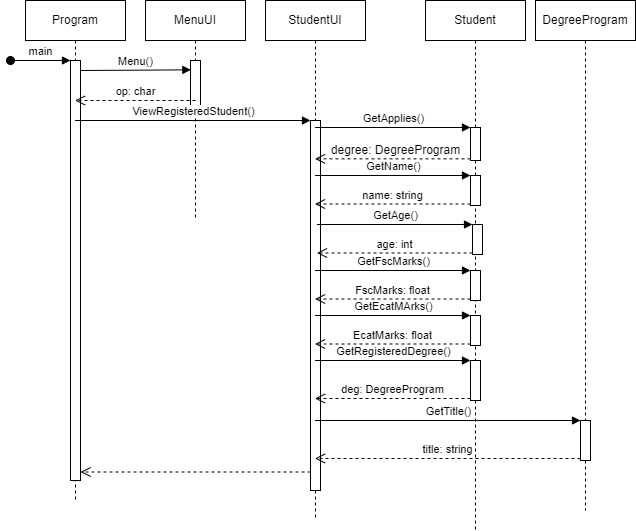
**Sequence Diagrams:**

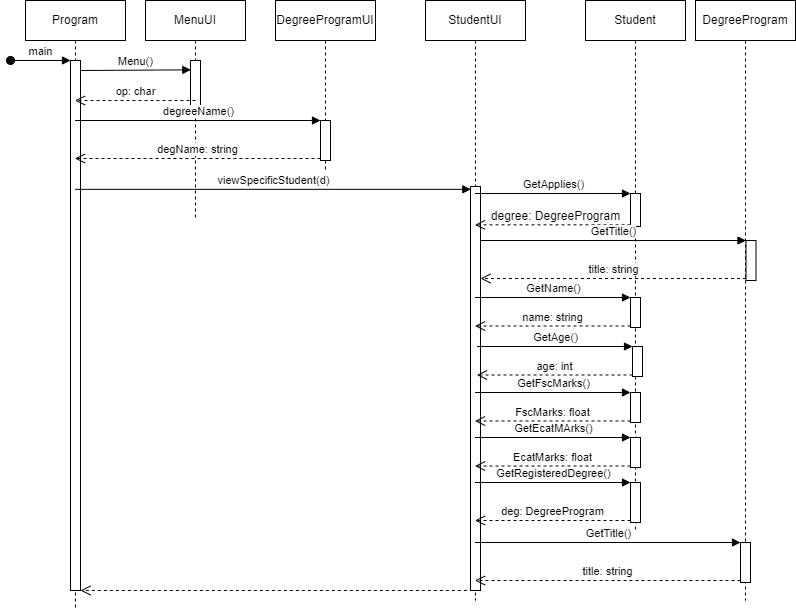
1. Add Student

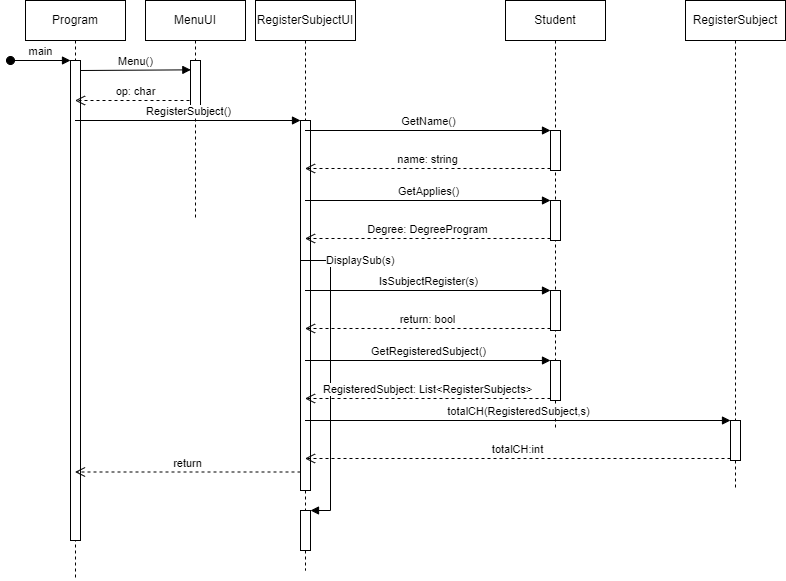


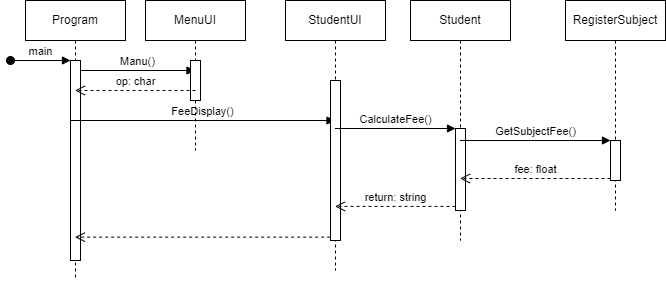
1. Add Degree
2. Generate Merit List



1. View Registered student
2. View Student Of Specific Degree



1. Register Subjects of Specific Student
2. Generate Fee



**Full Code**

**BL Classes:**

1. **DegreeProgram**

public DegreeProgram(string title, string duration, int seats)

{

this.title = title;

this.duration = duration;

this.seats = seats;

}

private string title;

private string duration;

private int seats;

private List<SubjectRegister> Contain = new List<SubjectRegister>();

public string GetTitle()

{

return title;

}

public string GetDuration()

{

return duration;

}

public int GetSeats()

{

return seats;

}

public void setSeats(int seats)

{

this.seats = seats;

}

public List<SubjectRegister> GetSubjctList()

{

return Contain;

}

public bool IsSubjectExists(SubjectRegister subject)

{

foreach (var sub in Contain)

{

if (sub.GetCode() == subject.GetCode())

{

return true;

}

}

return false;

}

public void AddSubjectsInList(List<SubjectRegister> Contain)

{

this.Contain = Contain;

}

public void AddSubject(SubjectRegister sub)

{

Contain.Add(sub);

}

1. **SubjectRegister**

public SubjectRegister(int CreditHours, string code,string SubjectType, int SubjectFee )

{

this.CreditHours = CreditHours;

this.code = code;

this.SubjectType = SubjectType;

this.SubjectFee = SubjectFee;

}

private int CreditHours;

private string code;

private string SubjectType;

private int SubjectFee;

public int GetCreditHours()

{

return CreditHours;

}

public string GetCode()

{

return code;

}

public string GetSubjectType()

{

return SubjectType;

}

public int GetSubjectFee()

{

return SubjectFee;

}

public int TotalCH(List<SubjectRegister> Subject, SubjectRegister s)

{

int totalCH = 0;

foreach(SubjectRegister sub in Subject)

{

totalCH = totalCH + sub.CreditHours;

}

totalCH = totalCH + s.CreditHours;

return totalCH;

}

1. **Student :**

public Student(string name, int age, float FscMarks, float EcatMarks, List<DegreeProgram> Prefrences)

{

this.name = name;

this.age = age;

this.FscMarks = FscMarks;

this.EcatMarks = EcatMarks;

Applies = null;

merit = Calculatemerit();

this.Register = Prefrences;

}

private string name;

private int age;

private float FscMarks;

private float EcatMarks;

private float merit;

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

private List<SubjectRegister> RegisteredSubject = new List<SubjectRegister>();

private DegreeProgram Applies;

public string GetName()

{

return name;

}

public int GetAge()

{

return age;

}

public float GetFscMarks()

{

return FscMarks;

}

public float GetEcatMarks()

{

return EcatMarks;

}

public float GetMerit()

{

return merit;

}

public DegreeProgram GetApplies()

{

return Applies;

}

public void setApplies(DegreeProgram Applies)

{

this.Applies = Applies;

}

public List<SubjectRegister> GetRegisteredSubject()

{

return RegisteredSubject;

}

public List<DegreeProgram> GetRegister()

{

return Register;

}

public float Calculatemerit()

{

float agg = ((FscMarks / 1100 \* 60) + (EcatMarks / 400 \* 40));

return agg;

}

public bool IsSubjectRegister(SubjectRegister sub)

{

foreach(SubjectRegister s in RegisteredSubject)

{

if(s == sub)

{

return true;

}

}

return false;

}

public string CalculateFee()

{

float fee = 0;

if (Applies != null)

{

for (int x = 0; x < RegisteredSubject.Count; x++)

{

fee = fee + RegisteredSubject[x].GetSubjectFee();

}

return ("Student name: " + name + " Student fee: " + fee);

}

return null;

}

}

**DL Classes:**

1. **DegreeProgramDL :**

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

public static DegreeProgram IsDegreeExists(string degTitle)

{

foreach (DegreeProgram d in degree)

{

if (d.GetTitle() == degTitle)

{

return d;

}

}

return null;

}

public static void AddDegreeInList(DegreeProgram deg)

{

degree.Add(deg);

}

public static void storeintoFile(string path, DegreeProgram d)

{

StreamWriter f = new StreamWriter(path, true);

string SubjectNames = "";

for (int x = 0; x < d.GetSubjctList().Count - 1; x++)

{

SubjectNames = SubjectNames + d.GetSubjctList()[x].GetSubjectType() + ";";

}

SubjectNames = SubjectNames + d.GetSubjctList()[d.GetSubjctList().Count - 1].GetSubjectType();

f.WriteLine(d.GetTitle() + "," + d.GetDuration() + "," + d.GetSeats() + "," + SubjectNames);

f.Flush();

f.Close();

}

public static bool readFromFile(string path)

{

StreamReader f = new StreamReader(path);

string record;

if (File.Exists(path))

{

while ((record = f.ReadLine()) != null)

{

string[] splittedRecord = record.Split(',');

string title = splittedRecord[0];

string duration = splittedRecord[1];

int seats = int.Parse(splittedRecord[2]);

string[] splittedRecordForSubject = splittedRecord[3].Split(';');

DegreeProgram d = new DegreeProgram(title, duration, seats);

for (int x = 0; x < splittedRecordForSubject.Length; x++)

{

SubjectRegister s = SubjectDL.isSubjectExists(splittedRecordForSubject[x]);

if (s != null)

{

d.AddSubject(s);

}

}

AddDegreeInList(d);

}

f.Close();

return true;

}

else

{

return false;

}

}

1. **SubjectDL:**

public static List<SubjectRegister> subjectList = new List<SubjectRegister>();

public static void addSubjectIntoList(SubjectRegister s)

{

subjectList.Add(s);

}

public static bool readFromFile(string path)

{

StreamReader f = new StreamReader(path);

string record;

if (File.Exists(path))

{

while ((record = f.ReadLine()) != null)

{

string[] splittedRecord = record.Split(',');

string code = splittedRecord[0];

string type = splittedRecord[1];

int CreditHours = int.Parse(splittedRecord[2]);

int subjectFees = int.Parse(splittedRecord[3]);

SubjectRegister s = new SubjectRegister(CreditHours, code, type, subjectFees);

addSubjectIntoList(s);

}

f.Close();

return true;

}

else

{

return false;

}

}

public static void storeintoFile(string path, SubjectRegister s)

{

StreamWriter f = new StreamWriter(path, true);

f.WriteLine(s.GetCode() + "," + s.GetSubjectType() + "," + s.GetCreditHours() + "," + s.GetSubjectFee());

f.Flush();

f.Close();

}

public static SubjectRegister isSubjectExists(string type)

{

foreach (SubjectRegister s in subjectList)

{

if (s.GetSubjectType() == type)

{

return s;

}

}

return null;

}

1. **StudentDL :**

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

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

public static List<Student> Sort()

{

newList = stu.OrderByDescending(o => o.GetMerit()).ToList();

return newList;

}

public static bool IsApplicable(Student s)

{

int seats = 0;

for (int x = 0; x < s.GetRegister().Count; x++)

{

DegreeProgram degree = s.GetRegister()[x];

if (degree.GetSeats() != 0)

{

seats = degree.GetSeats();

degree.setSeats(seats - 1);

s.setApplies(degree);

return true;

}

}

DegreeProgram Applies = s.GetApplies();

Applies = null;

s.setApplies(Applies);

return false;

}

public static void storeintoFile(string path, Student s)

{

StreamWriter f = new StreamWriter(path, true);

string degreeNames = "";

for (int x = 0; x < s.GetRegister().Count - 1; x++)

{

degreeNames = degreeNames + s.GetRegister()[x].GetTitle() + ";";

}

degreeNames = degreeNames + s.GetRegister()[s.GetRegister().Count - 1].GetTitle();

f.WriteLine(s.GetName() + "," + s.GetAge() + "," + s.GetFscMarks() + "," + s.GetEcatMarks() + "," + degreeNames);

f.Flush();

f.Close();

}

public static bool readFromFile(string path)

{

StreamReader f = new StreamReader(path);

string record;

if (File.Exists(path))

{

while ((record = f.ReadLine()) != null)

{

string[] splittedRecord = record.Split(',');

string name = splittedRecord[0];

int age = int.Parse(splittedRecord[1]);

float FscMarks = float.Parse(splittedRecord[2]);

float EcatMarks = float.Parse(splittedRecord[3]);

string[] splittedRecordForPreference = splittedRecord[4].Split(';');

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

for (int x = 0; x < splittedRecordForPreference.Length; x++)

{

DegreeProgram d = DegreeProgramDL.IsDegreeExists(splittedRecordForPreference[x]);

if (d != null)

{

if (!(preferences.Contains(d)))

{

preferences.Add(d);

}

}

}

Student s = new Student(name, age, FscMarks, EcatMarks, preferences);

stu.Add(s);

}

f.Close();

return true;

}

else

{

return false;

}

}

**UI Classes:**

1. **DegreeUI Class:**

public static void displayDegree(DegreeProgram d)

{

Console.WriteLine("Degree title: " + d.GetTitle());

}

public static List<DegreeProgram> AddPrefrences()

{

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

Console.WriteLine("All degrees");

foreach (DegreeProgram d in DegreeProgramDL.degree)

{

displayDegree(d);

}

Console.WriteLine("How many prefrences you want to add: ");

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

if (DegreeProgramDL.degree.Count >= p)

{

p = DegreeProgramDL.degree.Count;

}

else

{

Console.WriteLine("You can add prefrence only equal to No of degrees");

}

Console.WriteLine("Choose Your prefrences....");

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

{

Console.WriteLine("Enter your {0} prefrence: ", i + 1);

string pref = Console.ReadLine();

DegreeProgram d = DegreeProgramDL.IsDegreeExists(pref);

if (d != null)

{

Prefrences.Add(d);

}

else

{

Console.WriteLine("This degree is not available");

}

}

return Prefrences;

}

public static DegreeProgram AddDegree()

{

Console.WriteLine("Enter Degree name: ");

string title = Console.ReadLine();

Console.WriteLine("Enter Degree Duration: ");

string duration = Console.ReadLine();

Console.WriteLine("Enter seats : ");

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

DegreeProgram degree = new DegreeProgram(title, duration, seats);

List<SubjectRegister> Subject = SubjectUI.AddSubjects(degree);

degree.AddSubjectsInList(Subject);

return degree;

}

public static string degreeName()

{

Console.WriteLine("Enter degree name which students you want to see: ");

string d = Console.ReadLine();

return d;

}

1. **SubjectUI Class:**

public static List<SubjectRegister> AddSubjects(DegreeProgram degree)

{

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

Console.WriteLine("How many subjects you want to enter in this degree: ");

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

int totalCH = 0;

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

{

Console.WriteLine("Enter Subject code: ");

string code = (Console.ReadLine());

Console.WriteLine("Enter subject Type: ");

string SubjectType = Console.ReadLine();

Console.WriteLine("Enter subject Credit hours: ");

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

totalCH = totalCH + CreditHours;

Console.WriteLine("Enter subject fee: ");

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

SubjectRegister temp = new SubjectRegister(CreditHours, code, SubjectType, SubjectFee);

if (temp.TotalCH(Subject, temp) <= 20)

{

if (!degree.IsSubjectExists(temp))

{

Subject.Add(temp);

SubjectDL.storeintoFile("subject.txt", temp);

}

}

else

{

Console.WriteLine("A degree can only contain 20 credit hour");

}

}

return Subject;

}

public static void DisplaySub(SubjectRegister s)

{

Console.WriteLine("Subject Type is: " + s.GetSubjectType() + " Subject Code is: " + s.GetCode());

}

public static void RegisterSubject()

{

int idx = -1;

Console.WriteLine("Enter your name: ");

string name = Console.ReadLine();

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

{

if (StudentDL.stu[i].GetName() == name)

{

if (StudentDL.stu[i].GetApplies() != null)

{

idx = i;

break;

}

}

}

if (idx != -1)

{

foreach (SubjectRegister s in StudentDL.stu[idx].GetApplies().GetSubjctList())

{

DisplaySub(s);

Console.Write("Do you want to register this subject: press y for yes ");

char opt = char.Parse(Console.ReadLine());

if (opt == 'Y' || opt == 'y')

{

if (!StudentDL.stu[idx].IsSubjectRegister(s))

{

if (s.TotalCH(StudentDL.stu[idx].GetRegisteredSubject(), s) <= 9)

{

StudentDL.stu[idx].GetRegisteredSubject().Add(s);

}

else

{

Console.WriteLine("You can only registered subject less than or equal to 9 credit hours");

}

}

else

{

Console.WriteLine("Subject is already Registered");

}

}

}

1. **StudentUI Class:**

public static Student takeInput()

{

Console.WriteLine("Enter Student name: ");

string name = Console.ReadLine();

Console.WriteLine("Enter Your age: ");

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

Console.WriteLine("Enter Your FscMarks: ");

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

Console.WriteLine("Enter Your EcatMarks: ");

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

List<DegreeProgram> Prefrences = DegreeUI.AddPrefrences();

Student s = new Student(name, age, FscMarks, EcatMarks, Prefrences);

return s;

}

public static void viewRegisteredStudent()

{

foreach (Student s in StudentDL.stu)

{

if (s.GetApplies() != null)

{

Console.WriteLine("student name: " + s.GetName());

Console.WriteLine("student age: " + s.GetAge());

Console.WriteLine("student FscMarks: " + s.GetFscMarks());

Console.WriteLine("student ecatMarks: " + s.GetEcatMarks());

Console.WriteLine("student got admission in " + s.GetApplies().GetTitle());

Console.WriteLine(" ");

}

}

}

public static void viewSpecificStudent(string degree)

{

int count = 0;

Console.WriteLine("Students List of " + degree);

foreach (Student s in StudentDL.stu)

{

if (s.GetApplies() != null)

{

if (s.GetApplies().GetTitle() == degree)

{

Console.WriteLine("student name: " + s.GetName());

Console.WriteLine("student age: " + s.GetAge());

Console.WriteLine("student FscMarks: " + s.GetFscMarks());

Console.WriteLine("student ecatMarks: " + s.GetEcatMarks());

Console.WriteLine(" ");

count++;

}

}

}

if (count == 0)

{

Console.WriteLine("There are no students in this degree");

}

}

public static void DisplayFee()

{

foreach (var stu in StudentDL.stu)

{

string fee = stu.CalculateFee();

Console.WriteLine(fee);

}

}

public static void MeritList()

{

List<Student> newList = StudentDL.Sort();

foreach (Student s in newList)

{

if (StudentDL.IsApplicable(s) == true)

{

Console.WriteLine("Student " + s.GetName() + " got admission in: " + s.GetApplies().GetTitle());

}

else

{

Console.WriteLine("Student " + s.GetName() + " did'nt got admission");

}

}

}

1. **MenuUI:**

public static void header()

{

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

Console.WriteLine("\* UNIVERSITY ADMISSION MANAGEMENT SYSTEM \*");

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

Console.WriteLine(" ");

}

public static char menu()

{

char option;

header();

Console.WriteLine("////////// MAIN MENU /////////////// ");

Console.WriteLine(" ");

Console.WriteLine("Press 1 to Add student ");

Console.WriteLine("Press 2 to Add Degree Program ");

Console.WriteLine("Press 3 to generate Merit ");

Console.WriteLine("Press 4 to View Registered Student ");

Console.WriteLine("Press 5 to View Student of specific Program ");

Console.WriteLine("Press 6 to Register subject for specific Student ");

Console.WriteLine("Press 7 to Calculate fee for all registered student ");

Console.WriteLine("Press 8 to EXIT ");

Console.WriteLine("Enter your option ");

option = char.Parse(Console.ReadLine());

return option;

}

public static void systemCls()

{

Console.WriteLine("Press any key to continue....");

Console.ReadKey();

Console.Clear();

}

// end of system\_cls function

public static void DisplayWrongOption()

{

Console.WriteLine("You choose wrong option");

}

public static void Load()

{

Console.WriteLine("File Loaded Successfully");

}

public static void UnLoad()

{

Console.WriteLine("File does not Loaded Successfully");

}

public static void DisplayDegreeExist()

{

Console.WriteLine("Degree Already Exists");

}

**Driver Program:**

string degPath = "degree.txt";

string SubPath = "subject.txt";

string StuPath = "student.txt";

if (SubjectDL.readFromFile(SubPath))

{

MenuUI.Load();

}

else

{

MenuUI.UnLoad();

}

if (DegreeProgramDL.readFromFile(degPath))

{

MenuUI.Load();

}

else

{

MenuUI.UnLoad();

}

if (StudentDL.readFromFile(StuPath))

{

MenuUI.Load();

}

else

{

MenuUI.UnLoad();

}

Console.ReadKey();

while (true)

{

char option = MenuUI.menu();

MenuUI.systemCls();

MenuUI.header();

if (option == '1')

{

Student stu = StudentUI.takeInput();

StudentDL.stu.Add(stu);

StudentDL.storeintoFile(StuPath, stu);

}

else if (option == '2')

{

DegreeProgram d = DegreeUI.AddDegree();

if (DegreeProgramDL.IsDegreeExists(d.GetTitle()) == null)

{

DegreeProgramDL.AddDegreeInList(d);

DegreeProgramDL.storeintoFile(degPath, d);

}

else

{

MenuUI.DisplayDegreeExist();

}

}

else if (option == '3')

{

StudentUI.MeritList();

}

else if (option == '4')

{

StudentUI.viewRegisteredStudent();

}

else if (option == '5')

{

string d = DegreeUI.degreeName();

StudentUI.viewSpecificStudent(d);

}

else if (option == '6')

{

SubjectUI.RegisterSubject();

}

else if (option == '7')

{

StudentUI.DisplayFee();

}

else if (option == '8')

{

break;

}

else

{

MenuUI.DisplayWrongOption();

}

MenuUI.systemCls();

}

**Scenerio4: CofeeShop**

**Case Study Scenario:**

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:**

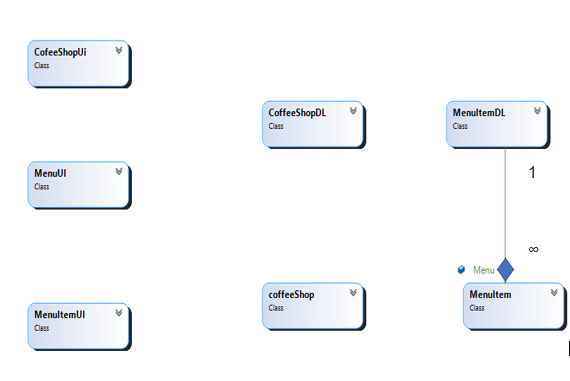
1. Domain Model with classes name.



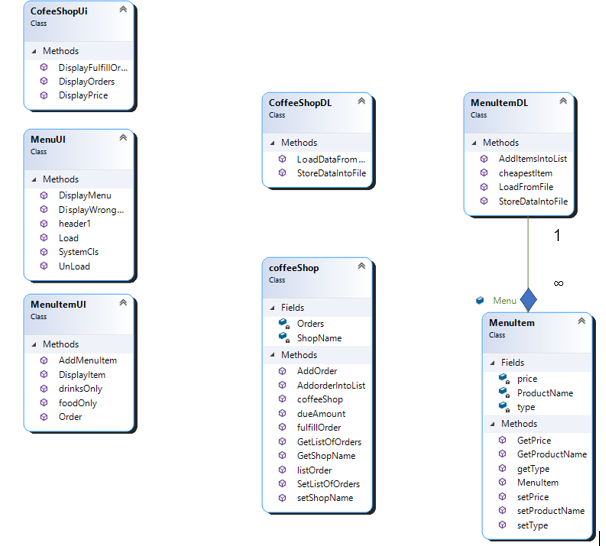
2. Domain Model with relations and Constraints.



3. Domain Model with multiplicity

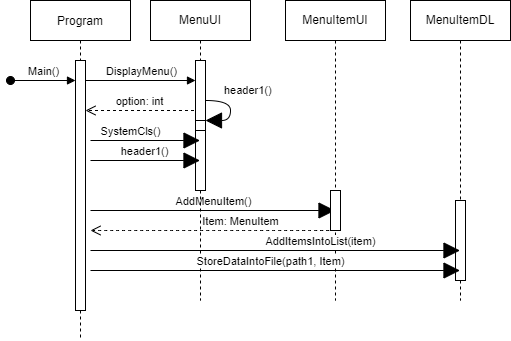


**Class Diagram:**

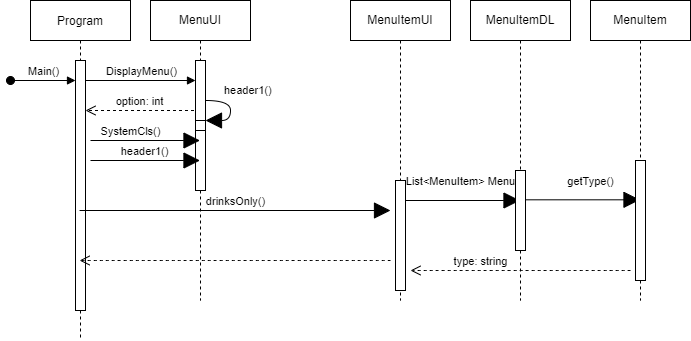


**Sequence Diagrams:**

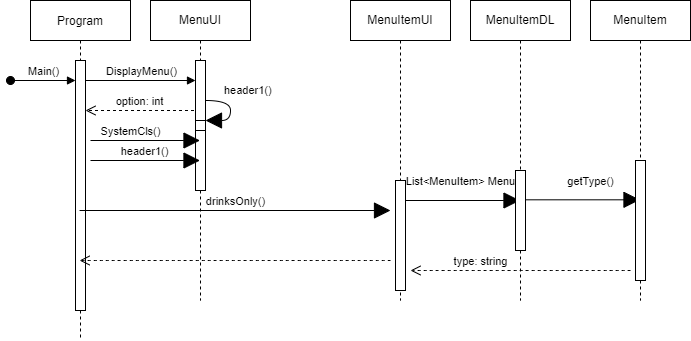
1. Add Menu Item



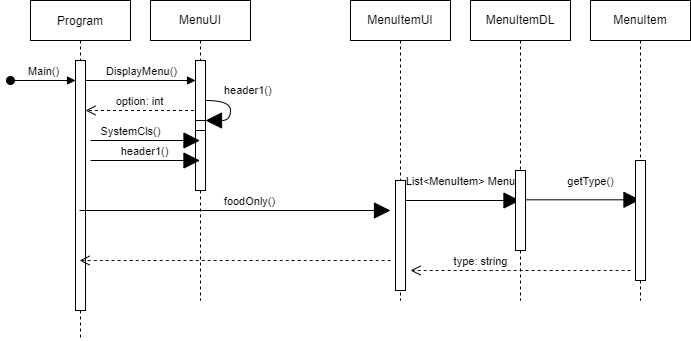
1. View the cheapest Item in Menu



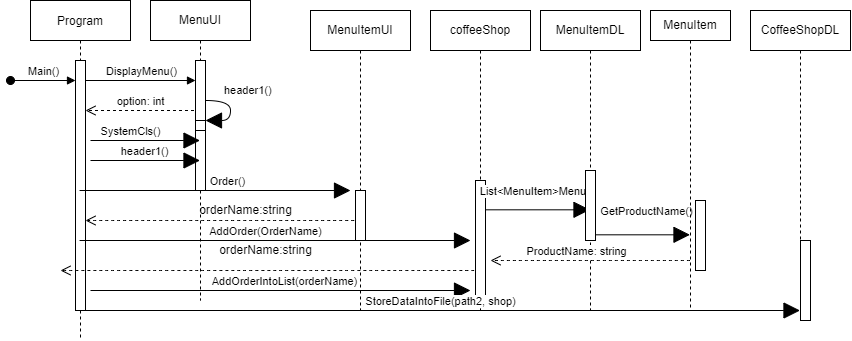
1. View the drinks menu



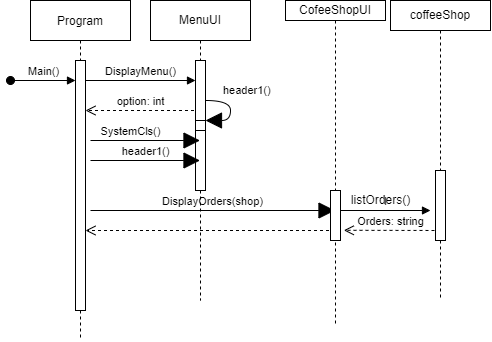
1. View the Food menu



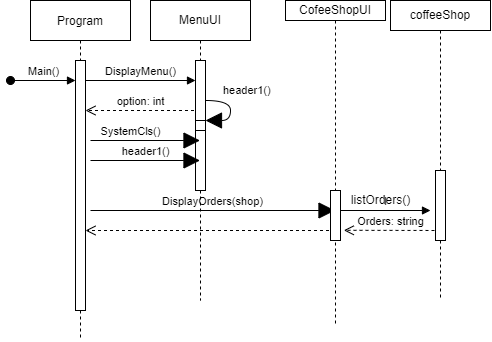
1. Place Order



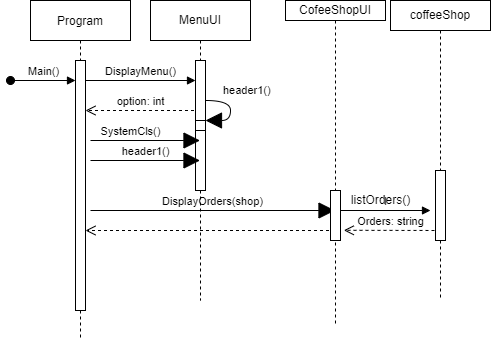
1. fulfill Order



1. View Order’s List



1. Total Payble Amount



**Full Code**

**BL Classes:**

1. **MenuItem**

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

{

this.ProductName = ProductName;

this.type = type;

this.price = price;

}

private string ProductName;

private string type;

private float price;

public string GetProductName()

{

return ProductName;

}

public string getType()

{

return type;

}

public float GetPrice()

{

return price;

}

public void setProductName(string ProductName)

{

this.ProductName = ProductName;

}

public void setType(string type)

{

this.type = type;

}

public void setPrice(float price )

{

this.price = price;

}

1. **Coffee Shop**

public coffeeShop(string ShopName)

{

this.ShopName = ShopName;

}

private string ShopName;

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

public string AddOrder(string OrderName)

{

foreach (MenuItem item in MenuItemDL.Menu)

{

if (OrderName == item.GetProductName())

{

return OrderName;

}

}

return "This Item is Currently Unavailable";

}

public void AddorderIntoList(string name)

{

Orders.Add(name);

}

public string GetShopName()

{

return ShopName;

}

public void setShopName(string ShopName)

{

this.ShopName = ShopName;

}

public List<string> GetListOfOrders()

{

return Orders;

}

public void SetListOfOrders(List<string>Orders)

{

this.Orders = Orders;

}

public string fulfillOrder()

{

if (Orders.Count != 0)

{

string name = Orders[0];

Orders.RemoveAt(0);

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

}

return "All orders have been fulfilled ";

}

public List<string> listOrder()

{

if (Orders.Count != 0)

{

return Orders;

}

return null;

}

public double dueAmount()

{

double price = 0;

foreach (string name in Orders)

{

foreach (MenuItem item in MenuItemDL.Menu)

{

if (name == item.GetProductName())

{

price += item.GetPrice();

}

}

}

return price;

}

**DL Classes:**

1. **MenuItemDL:**

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

public static void AddItemsIntoList(MenuItem item)

{

Menu.Add(item);

}

public static string cheapestItem()

{

List<MenuItem> SortedMenu = Menu.OrderBy(o => o.GetPrice()).ToList();

return SortedMenu[0].GetProductName();

}

public static void StoreDataIntoFile(string path, MenuItem item)

{

StreamWriter file = new StreamWriter(path, true);

file.WriteLine(item.GetProductName() + "," + item.getType() + "," + item.GetPrice());

file.Flush();

file.Close();

}

public static bool LoadFromFile(string path)

{

StreamReader file = new StreamReader(path);

if(File.Exists(path))

{

string record;

while ((record = file.ReadLine()) != null)

{

string[] splitRecord = record.Split(',');

string ProductName = splitRecord[0];

string type = splitRecord[1];

float price = float.Parse(splitRecord[2]);

MenuItem item = new MenuItem(ProductName, type, price);

AddItemsIntoList(item);

}

file.Close();

return true;

}

return false;

}

1. **CofeeShopDL:**

public static void StoreDataIntoFile(string path, coffeeShop shop)

{

StreamWriter file = new StreamWriter(path);

foreach (string orders in shop.GetListOfOrders())

{

file.WriteLine(orders);

}

file.Flush();

file.Close();

}

public static bool LoadDataFromFile(string path, coffeeShop shop)

{

StreamReader file = new StreamReader(path, false);

if (File.Exists(path))

{

string record;

while ((record = file.ReadLine()) != null)

{

shop.AddOrder(record);

shop.AddorderIntoList(record);

}

file.Close();

return true;

}

return false;

}

**UI Classes:**

1. **MenuItemUI:**

public static void DisplayItem()

{

foreach (MenuItem item in MenuItemDL.Menu)

{

Console.WriteLine("Item name: " + item.GetProductName());

Console.WriteLine("Item Type " + item.getType());

Console.WriteLine("item Price " + item.GetPrice());

Console.WriteLine("\n");

}

}

public static void drinksOnly()

{

foreach (MenuItem item in MenuItemDL.Menu)

{

if (item.getType() == "Drink" || item.getType() == "drink")

{

Console.WriteLine("Item name: " + item.GetProductName());

Console.WriteLine("Item Type " + item.getType());

Console.WriteLine("item Price " + item.GetPrice());

Console.WriteLine("\n");

}

}

}

public static void foodOnly()

{

foreach (MenuItem item in MenuItemDL.Menu)

{

if (item.getType() == "Food" || item.getType() == "food")

{

Console.WriteLine("Item name: " + item.GetProductName());

Console.WriteLine("Item Type " + item.getType());

Console.WriteLine("item Price " + item.GetPrice());

Console.WriteLine("\n");

}

}

}

public static MenuItem AddMenuItem()

{

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

string ProductName = Console.ReadLine();

Console.WriteLine("Enter type of item ");

string type = Console.ReadLine();

Console.WriteLine("Enter price of item ");

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

MenuItem Item = new MenuItem(ProductName, type, price);

return Item;

}

public static string Order()

{

Console.WriteLine("Enter your order please: ");

string OrderName = Console.ReadLine();

return OrderName;

}

1. **CoffeeShopUI:**

public static void DisplayPrice(double price)

{

Console.WriteLine("your total amount is: " + price);

}

public static void DisplayOrders(coffeeShop shop)

{

List<string> orders = shop.listOrder();

if (orders != null)

{

foreach (string order in orders)

{

Console.WriteLine(order);

}

}

}

public static void DisplayFulfillOrders(coffeeShop shop)

{

Console.WriteLine(shop.fulfillOrder());

}

1. **MenuUI :**

public static void SystemCls()

{

Console.WriteLine("Press any key to continue....");

Console.ReadKey();

Console.Clear();

}

public static int DisplayMenu()

{

header1();

Console.WriteLine("//////// Main Menu //////////// ");

Console.WriteLine("Select one of the following options number . . .");

Console.WriteLine("1. Add a menu Item ");

Console.WriteLine("2. View the cheapest item in menu");

Console.WriteLine("3. View the Drink's menu");

Console.WriteLine("4. View the Food's menu");

Console.WriteLine("5. Place Order ");

Console.WriteLine("6. Fulfill the order ");

Console.WriteLine("7. View the Order's List ");

Console.WriteLine("8. Total payable amount ");

Console.WriteLine("9. Exit");

Console.Write("Choose your option: ");

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

return option;

}

public static void DisplayWrongOption()

{

Console.WriteLine("You choose wrong option");

}

public static void Load()

{

Console.WriteLine("File loaded Successfully");

Console.ReadKey();

}

public static void UnLoad()

{

Console.WriteLine("File does not loaded Successfully");

Console.ReadKey();

Console.Clear();

}

**Driver Program:**

static void Main(string[] args)

{

string path1 = "Items.txt";

string path2 = "Orders.txt";

coffeeShop shop = new coffeeShop("Tesha's Coffee Shop: ");

if (MenuItemDL.LoadFromFile(path1))

{

MenuUI.Load();

}

else

{

MenuUI.UnLoad();

}

if (CoffeeShopDL.LoadDataFromFile(path2, shop))

{

MenuUI.Load();

}

else

{

MenuUI.UnLoad();

}

do

{

MenuUI.header1();

int option = MenuUI.DisplayMenu();

MenuUI.SystemCls();

MenuUI.header1();

if (option == 1)

{

MenuItem item = MenuItemUI.AddMenuItem();

MenuItemDL.AddItemsIntoList(item);

MenuItemDL.StoreDataIntoFile(path1, item);

}

else if (option == 2)

{

Console.WriteLine(MenuItemDL.cheapestItem());

}

else if (option == 3)

{

MenuItemUI.drinksOnly();

}

else if (option == 4)

{

MenuItemUI.foodOnly();

}

else if (option == 5)

{

string OrderName = MenuItemUI.Order();

string order = shop.AddOrder(OrderName);

if (order != "This Item is Currently Unavailable")

{

shop.AddorderIntoList(order);

CoffeeShopDL.StoreDataIntoFile(path2, shop);

}

else

{

Console.WriteLine(order);

}

}

else if (option == 6)

{

CofeeShopUi.DisplayFulfillOrders(shop);

CoffeeShopDL.StoreDataIntoFile(path2, shop);

}

else if (option == 7)

{

CofeeShopUi.DisplayOrders(shop);

}

else if (option == 8)

{

double price = shop.dueAmount();

CofeeShopUi.DisplayPrice(price);

}

else if (option == 9)

{

break;

}

else

{

MenuUI.DisplayWrongOption();

}

MenuUI.SystemCls();

}

while (true);

}

}

}

**Scenerio5: Point and Line**

**Case Study Scenario:**

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.

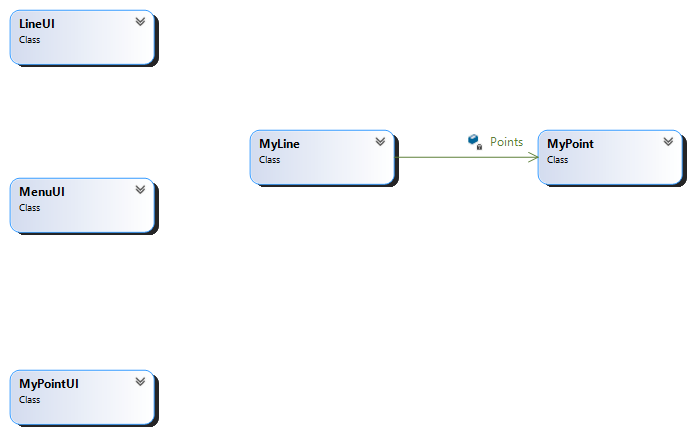
Use distance formula to calculate the length of the line

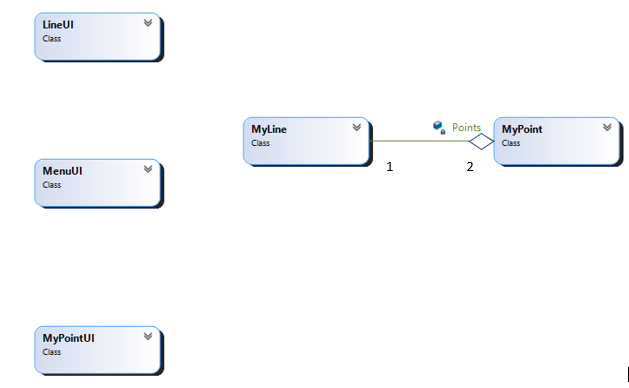
**Domain Model:**

1. Domain Model with classes name.

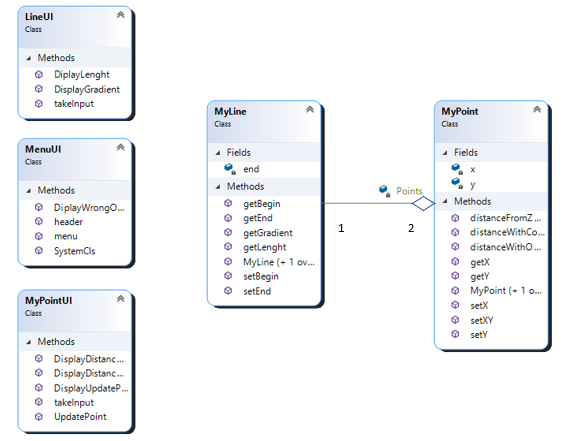


2. Domain Model with relations and Constraints.



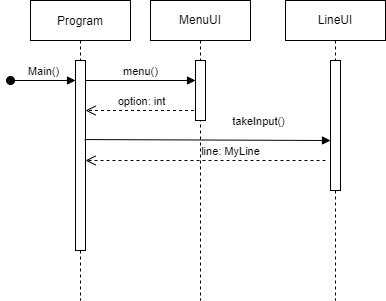
3.Domain Model with multiplicity

**Class Diagram:**

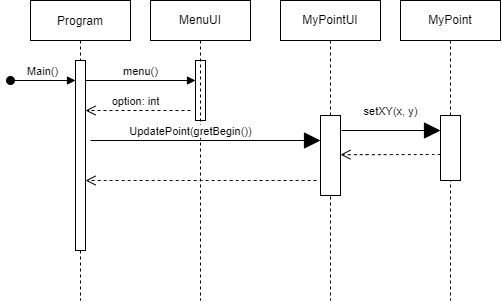


**Sequence Diagrams:**

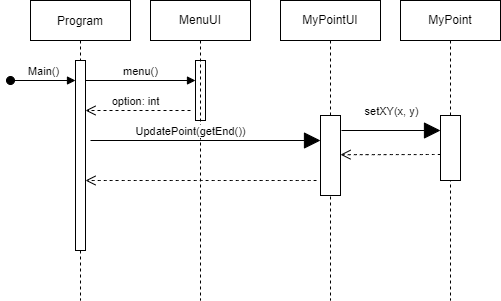
1. Make a line



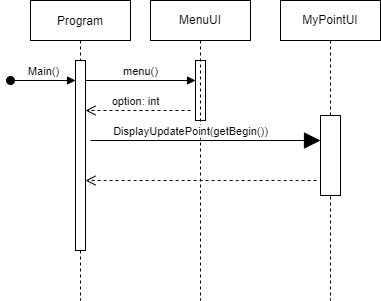
1. Update the begin point



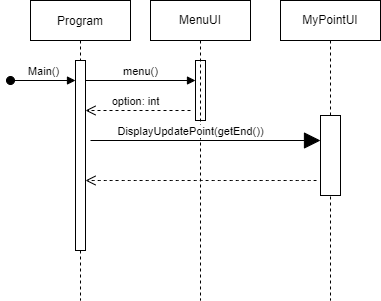
1. Update end points



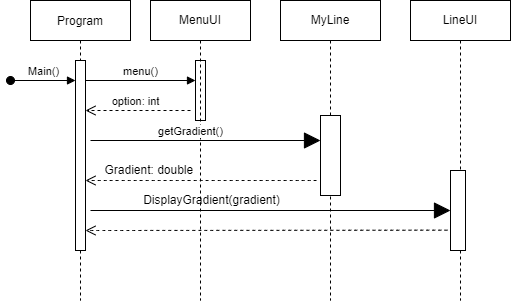
1. Show the begin point



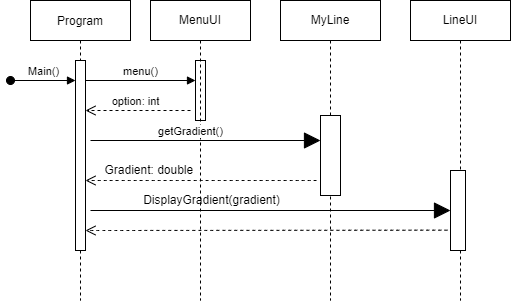
1. Show the end points



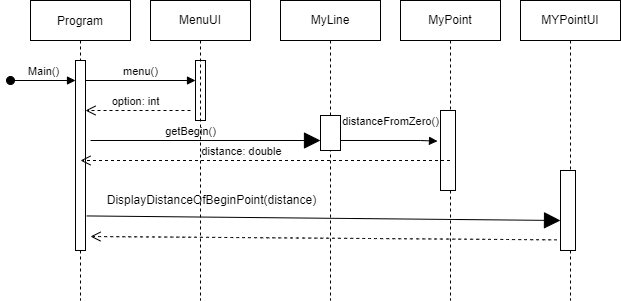
1. Get the length of line

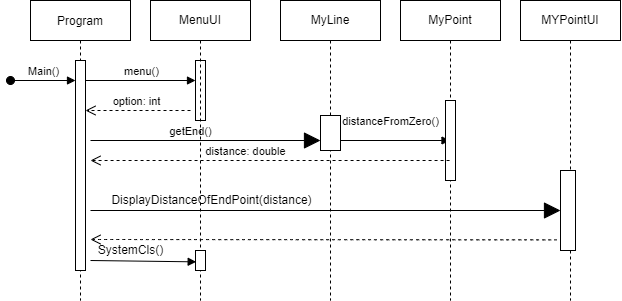


1. Get the Gradient of line



1. Distance of begin point with zero coordinates



1. Distance of end point with zero coordinates

**Full Code**

**BL Classes:**

1. **MyPoint**

public MyPoint(int x, int y)

{

this.x=x;

this.y=y;

}

public MyPoint()

{

x = 0;

y = 0;

}

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 distanceWithCord(int x, int y)

{

return Math.Sqrt(Math.Pow((this.x-x),2)+ Math.Pow((this.y - y), 2)) ;

}

public double distanceWithObject(MyPoint anotherPoint)

{

return Math.Sqrt(Math.Pow((this.x - anotherPoint.x), 2) + Math.Pow((this.y - anotherPoint.y), 2));

}

public double distanceFromZero()

{

return Math.Sqrt(Math.Pow(this.x, 2) + Math.Pow(this.y, 2));

}

1. **MyLine**

public 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 getLenght()

{

return Math.Sqrt(Math.Pow((begin.getX() - end.getX()), 2) + Math.Pow((begin.getY() - end.getY()), 2));

}

public double getGradient()

{

return (end.getY() - begin.getY()) / (end.getX() - begin.getX());

}

**UI Classes:**

1. **MyPointUI:**

public static MyPoint takeInput()

{

Console.Write("Enter x coordinate : ");

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

Console.Write("Enter y coordinate : ");

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

MyPoint point = new MyPoint(x, y);

return point;

}

public static void UpdatePoint(MyPoint UpdatePoint)

{

Console.Write("Enter x coordinate : ");

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

Console.Write("Enter y coordinate : ");

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

UpdatePoint.setXY(x, y);

}

public static void DisplayUpdatePoint(MyPoint UpdatePoint)

{

Console.WriteLine("x coordinate of line " + UpdatePoint.getX());

Console.WriteLine("y coordinate of line " + UpdatePoint.getY());

}

public static void DisplayDistanceOfBeginPoint(double Distance)

{

Console.WriteLine("distance of begin point from zero coordinate: " + Distance);

}

public static void DisplayDistanceOfEndPoint(double Distance)

{

Console.WriteLine("distance of end point from zero coordinate: " + Distance);

}

1. **LineUI:**

public static MyLine takeInput()

{

Console.WriteLine("Enter coordinates for begining: \n");

MyPoint begin = MyPointUI.takeInput();

Console.WriteLine("\n");

Console.WriteLine("Enter coordinates for ending: \n");

MyPoint end = MyPointUI.takeInput();

MyLine line = new MyLine(begin, end);

return line;

}

public static void DiplayLenght(double Lenght)

{

Console.WriteLine("Lenght of line is: " + Lenght);

}

public static void DisplayGradient(double Gradient)

{

Console.WriteLine("Gradient of line is: " + Gradient);

}

1. **MenuUI:**

public static void header()

{

Console.WriteLine("////////////////////////////////////////////////////////////");

Console.WriteLine("-- POINT AND LINE APPLICATION --");

Console.WriteLine("////////////////////////////////////////////////////////////");

Console.WriteLine(" ");

}

// end of 2nd header function

public static void SystemCls()

{

Console.WriteLine("Press any key to continue....");

Console.ReadKey();

Console.Clear();

}

public static int menu()

{

header();

Console.WriteLine("//////// Main Menu //////////// ");

Console.WriteLine("Select one of the following options number . . .");

Console.WriteLine("1. Make a line ");

Console.WriteLine("2. Update the begin point");

Console.WriteLine("3. Update the end point");

Console.WriteLine("4. Show the begin point");

Console.WriteLine("5. show the end point");

Console.WriteLine("6. Get the lenght of the line");

Console.WriteLine("7. Get the gradient of the line ");

Console.WriteLine("8. Find the distance of begin point from zero coordinates ");

Console.WriteLine("9. Find the distance of end point from zero coordinates ");

Console.WriteLine("10. Exit");

Console.Write("Choose your option: ");

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

SystemCls();

return option;

}

public static void DiplayWrongOption()

{

Console.WriteLine("you choose wrong option");

}

**Driver Program:**

MyLine line=new MyLine();

do

{

Console.Clear();

int option = MenuUI.menu();

if (option == 1)

{

line = LineUI.takeInput();

}

else if (option == 2)

{

MyPointUI.UpdatePoint(line.getBegin());

}

else if (option == 3)

{

MyPointUI.UpdatePoint(line.getEnd());

}

else if (option == 4)

{

MyPointUI.DisplayUpdatePoint(line.getBegin());

}

else if (option == 5)

{

MyPointUI.DisplayUpdatePoint(line.getEnd());

}

else if (option == 6)

{

double Lenght = line.getLenght();

LineUI.DiplayLenght(Lenght);

}

else if (option == 7)

{

double Gradient = line.getGradient();

LineUI.DisplayGradient(Gradient);

}

else if (option == 8)

{

double Distance= line.getBegin().distanceFromZero();

MyPointUI.DisplayDistanceOfBeginPoint(Distance);

}

else if (option == 9)

{

double Distance = line.getEnd().distanceFromZero();

MyPointUI.DisplayDistanceOfEndPoint(Distance);

}

else if (option == 10)

{

break;

}

else

{

MenuUI.DiplayWrongOption();

}

MenuUI.SystemCls();

} while (true);

**Scenerio6: GameObject**

Case Study Scenario:

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

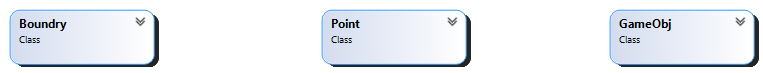
one step toward right.

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

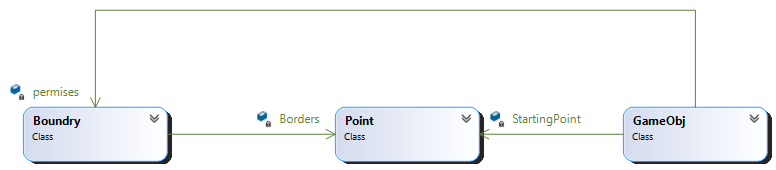
○ Draw: When called, this method will draw the shape on the console.

**Domain Model:**

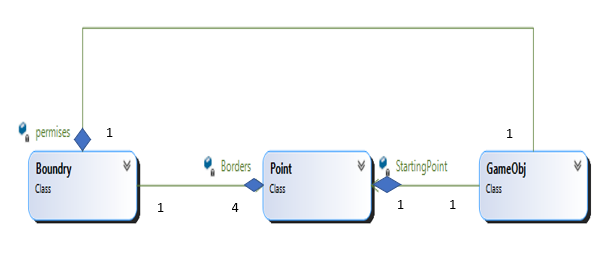
1. Domain Model with classes name.



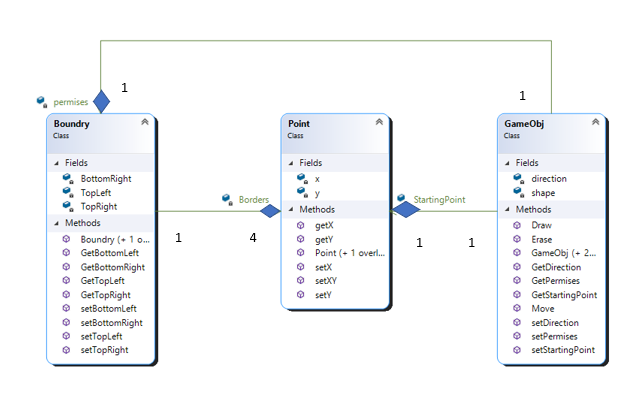
2. Domain Model with relations and Constraints.



3.Domain Model with multiplicity



**Class Diagram:**



**Full Code**

**BL Classes:**

1. **Point**

public MyPoint(int x, int y)

{

this.x=x;

this.y=y;

}

public MyPoint()

{

x = 0;

y = 0;

}

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 distanceWithCord(int x, int y)

{

return Math.Sqrt(Math.Pow((this.x-x),2)+ Math.Pow((this.y - y), 2)) ;

}

public double distanceWithObject(MyPoint anotherPoint)

{

return Math.Sqrt(Math.Pow((this.x - anotherPoint.x), 2) + Math.Pow((this.y - anotherPoint.y), 2));

}

public double distanceFromZero()

{

return Math.Sqrt(Math.Pow(this.x, 2) + Math.Pow(this.y, 2));

}

1. **Boundry**

public Boundry()

{

TopLeft = new Point();

TopRight = new Point(0, 90);

BottomLeft = new Point(90, 0);

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 Point GetTopLeft()

{

return TopLeft;

}

public Point GetTopRight()

{

return TopRight;

}

public Point GetBottomLeft()

{

return BottomLeft;

}

public Point GetBottomRight()

{

return BottomRight ;

}

public void setTopLeft(Point TopLeft)

{

this.TopLeft = TopLeft; ;

}

public void setTopRight(Point TopRight)

{

this.TopRight = TopRight; ;

}

public void setBottomLeft(Point BottomLeft)

{

this.BottomLeft = BottomLeft;

}

public void setBottomRight(Point BottomRight)

{

this.BottomRight = BottomRight;

}

1. **Game Object**

public GameObj()

{

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

StartingPoint = new Point();

permises = new Boundry();

direction = "LeftToRight";

}

public GameObj(char[,]shape, Point StartingPoint)

{

this.shape = shape;

this.StartingPoint = StartingPoint;

permises = new Boundry();

direction = "LeftToRight";

}

public GameObj(char[,]shape, Point StartingPoint, Boundry permises, string direction)

{

this.shape = shape;

this.StartingPoint = StartingPoint;

this.permises = permises;

this.direction = direction;

}

private char[,] shape;

private Point StartingPoint;

private Boundry permises;

private string direction;

private int PatrolCount=1;

private int projectileCount;

public Point GetStartingPoint()

{

return StartingPoint;

}

public Boundry GetPermises()

{

return permises;

}

public string GetDirection()

{

return direction;

}

public void setStartingPoint(Point StartingPoint)

{

this.StartingPoint = StartingPoint ;

}

public void setPermises(Boundry permises)

{

this.permises= permises;

}

public void setDirection(string direction)

{

this.direction= direction;

}

public void Erase()

{

for (int i = StartingPoint.getX(); i < StartingPoint.getX() + 5; i++)

{

for (int j = StartingPoint.getY(); j < StartingPoint.getY() + 3; j++)

{

Console.SetCursorPosition(j, i);

Console.Write(" ");

}

}

}

public void Draw()

{

int x = 0, y = 0;

for (int i = StartingPoint.getX(); i < StartingPoint.getX() + 5; i++ )

{

for(int j= StartingPoint.getY(); j < StartingPoint.getY()+3; j++)

{

Console.SetCursorPosition(j, i);

Console.Write(shape[x, y]);

y++;

}

x++;

y = 0;

}

}

public void Move()

{

if(direction == "LeftToRight")

{

int startPoint;

if(StartingPoint.getY() != 90)

{

startPoint= StartingPoint.getY();

StartingPoint.setY(startPoint+1);

}

}

else if(direction == "RightToLeft")

{

int startPoint;

if (StartingPoint.getY() != 0)

{

startPoint = StartingPoint.getY();

StartingPoint.setY(startPoint - 1);

}

}

else if(direction == "Diagonal")

{

int xPoint;

int yPoint;

if(StartingPoint.getY() != 90 || StartingPoint.getY() != 90)

{

yPoint= StartingPoint.getY();

xPoint= StartingPoint.getX();

StartingPoint.setY(yPoint + 1);

StartingPoint.setX(xPoint + 1);

}

}

else if (direction == "Patrol")

{

if (PatrolCount == 1)

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

if (PatrolCount == 2)

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

if (StartingPoint.getY() == permises.GetBottomLeft().getY())

PatrolCount = 2;

if (StartingPoint.getY() == permises.GetBottomRight().getY())

PatrolCount = 1;

}

else if (direction == "Projectile")

{

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

{

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

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

}

if (projectileCount == 6 || projectileCount == 7)

{

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

}

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

{

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

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

}

projectileCount++;

if(projectileCount == 11)

{

projectileCount = 0;

}

}

**Driver Program:**

static void Main(string[] args)

{

char[,] triangle = new char[5, 3] { { '@', ' ', ' ' }, { '@', '@', ' ' }, { '@', '@', '@' }, { '@', '@', ' ' }, { '@', ' ', ' ' } };

char[,] optriangle = new char[5, 3] { { ' ', ' ', '@' }, { ' ', '@', '@' }, { '@', '@', '@' }, { ' ', '@', '@' }, { ' ', ' ', '@' } };

Boundry b = new Boundry(new Point(0, 0), new Point(0, 90), new Point(90, 0), new Point(90, 90));

GameObj g1 = new GameObj(triangle, new Point(5, 5), b, "LeftToRight");

GameObj g2 = new GameObj(optriangle, new Point(30, 60), b, "RightToLeft");

List<GameObj> list = new List<GameObj>();

list.Add(g1);

list.Add(g2);

while(true)

{

Thread.Sleep(100);

foreach(GameObj g in list)

{

g.Erase();

g.Move();

g.Draw();

}

}