TermFinal Preperation- OOP(C#)

Delegate

```
//delegate returnType delegateName(parameters);
delegate void MyDelegate(int x, int y);

//deleegateName variableName = function
MyDelegate plus = (x, y) => Console.WriteLine(x + y);
MyDelegate minus = (x, y) => Console.WriteLine(x - y);
MyDelegate mul = (x, y) => Console.WriteLine(x * y);
MyDelegate div = (x, y) => Console.WriteLine(x / y);
//deleegateName variableName=functionName
MyDelegate  sh=show;
plus += minus;//multicast delegates,now this will call plus,then minus.
plus(10,12);
/*output:
22
-2
*/
sh(444,1145);

static void show(int x,int y){
    Console.WriteLine(x+" "+y);
}
```

Enum

```
    enum Operations{plus, minus, mul, div}
    string s=Console.ReaddLine();
    if(Enum.TryParse<Operations>(s,true,out Operations op)){
        Console.Write($"successful {op}");
    }
```

Abstract

- Abstract class may contain abstruct or non-abstract methods.
- Abstract methods must be within a abstract class.
- Abstract methods can't be private.
- No instance creation is allowed for abstract class.
- Abstract methods don't have a body.
- Child class must override abstract methods.

Method Chaining

```
using System;
namespace Termfinal{
    class Person{
        public string Name{get;private set;}
        public int Age{get;private set;}
        public Person SetName(string name){
            Name=name;
            return this;
        }
        public Person SetAge(int age){
            Age=age;
            return this;
        }
        public override string ToString()
        {
            return string.Join("\n",$"Name: {Name}",$"Age : {Age}");
        }
    }
    class Program{
        static void Main(){
            //method chaining
            Person person=(new Person()).SetName("Cristiano").SetAge(39);
            Console.WriteLine(person);
        }
    }
}
```

Vehicle

```
using System;

class Vehicle
{
    public string Make { get; set; }
    public string Model { get; set; }
    public int Year { get; set; }
    public virtual void ShowInfo()
    {
        Console.WriteLine("Make: " + Make);
        Console.WriteLine($"Model: {Model}");
        Console.WriteLine($"Year: {Year}");
    }
}

class Car : Vehicle
{
    public int NumberOfDoors { get; set; }
    public override void ShowInfo()
    {
        base.ShowInfo();
        Console.WriteLine($"Number of Doors: {NumberOfDoors}");
    }
}

class Motorcycle : Vehicle
{
    public bool HasSidecar { get; set; }
    public override void ShowInfo()
    {
        base.ShowInfo();
        Console.WriteLine($"Has Sidecar: {HasSidecar}");
    }
}

class Program
{
    static void Main()
    {
        Car car = new Car();
        car.Make = "Corolla";
        car.Model = "X50";
        car.Year = 2015;
        car.NumberOfDoors = 4;

        Motorcycle motorcycle = new Motorcycle()
        {
            Make = "Honda",
            Model = "S12",
            Year = 2012,
            HasSidecar = false
        };

        motorcycle.ShowInfo();
        car.ShowInfo();
    }
}
```

Library System

```
using System;
using System.Collections;
namespace Termfinal
{
    class LibraryItem
    {
        public string Title { get; set; }
        public string Author { get; set; }
        public int Year { get; set; }
        public bool CheckOut { get; set; }
        public virtual void PrintDetails()
        {
            Console.WriteLine($"Title : {Title}");
            Console.WriteLine($"Author : {Author}");
            Console.WriteLine($"Year : {Year}");
            Console.WriteLine($"CheckOut : {CheckOut}");
        }
    }
    class Book : LibraryItem
    {
        public int Pages { get; set; }
        public override void PrintDetails()
        {
            base.PrintDetails();
            Console.WriteLine($"Pages: {Pages}");
        }
    }
    class DVD : LibraryItem
    {
        public double Duration { get; set; }
        public override void PrintDetails()
        {
            base.PrintDetails();
            Console.WriteLine($"Duration : {Duration}");
        }
    }
    class Library : IEnumerable<LibraryItem>
    {
        List<LibraryItem> libraryItems = new List<LibraryItem>();
        public IEnumerator<LibraryItem> GetEnumerator() => libraryItems.GetEnumerator();
        IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();

        public void Add(LibraryItem libraryItem) => libraryItems.Add(libraryItem);
        public void Remove(LibraryItem libraryItem) => libraryItems.Remove(libraryItem);
        private void Display(IEnumerable<LibraryItem> libraryItems)
        {
            foreach (var v in libraryItems) v.PrintDetails();
        }
        public void DisplayAll()
        {
            Display(libraryItems);
        }
        public void DisplayAvailablItems()
        {
            Display(libraryItems.Where(item=> !item.CheckOut));
        }
        public void DisplayCheckedoutlItems()
        {
            Display(libraryItems.Where(item=> item.CheckOut));
        }
    }
    public class Program{
        public static void Main(){
            Book book1=new Book(){Title="ABC",Author="X",Year=2000,CheckOut=false,Pages=100};
            Book book2=new Book(){Title="ABD",Author="Y",Year=2015,CheckOut=true,Pages=510};
            DVD dVD1=new DVD(){Title="DVD1",Author="X1",Year=2011,CheckOut=true,Duration=68};
            DVD dVD2=new DVD(){Title="DVD2",Author="X2",Year=2021,CheckOut=false,Duration=59};
            Library libraryItems=new Library();
            libraryItems.Add(book1);
            libraryItems.Add(book2);
            libraryItems.Add(dVD1);
            libraryItems.Add(dVD2);
            libraryItems.DisplayAll();
            libraryItems.DisplayAvailablItems();
            libraryItems.DisplayCheckedoutlItems();
        }
    }
}
```

Zoo System

```
using System;
namespace TermFinal
{
    interface IAnimalInfo{
        void PrintInfo();

    }
    abstract class Animal{
        public string Name{get;set;}
        public int Age{get;set;}
        public abstract void MakeSound();
        protected void ShowInfo(){
            Console.WriteLine($"Name: {Name}");
            Console.WriteLine($"Age: {Age}");
        }
    }
    class Lion : Animal,IAnimalInfo
    {

        public override void MakeSound()
        {
            Console.WriteLine($"Roar!");
        }

        void IAnimalInfo.PrintInfo()
        {
            ShowInfo();
            //Console.WriteLine($"MakeSound : {}")
        }
    }
    class Elephant:Animal,IAnimalInfo{
        public override void MakeSound()
        {
            Console.WriteLine($"Trumpet!");
        }

        void IAnimalInfo.PrintInfo()
        {
            ShowInfo();
        }
    }
    class Monkey : Animal, IAnimalInfo
    {
        public override void MakeSound()
        {
            Console.WriteLine($"Ooh-Ooh Ah-Ah!");

        }

        void IAnimalInfo.PrintInfo()
        {
            ShowInfo();
        }
    }
    class Program{
        public static void Main(){
            Lion lion1=new Lion(){Name="L1",Age=5};
            lion1.MakeSound();
            ((IAnimalInfo)lion1).PrintInfo();
            Monkey monkey1=new Monkey(){Name="M1",Age=1};
            monkey1.MakeSound();
            ((IAnimalInfo)monkey1).PrintInfo();
            Elephant elephant1=new Elephant(){Name="E1",Age=7};
            elephant1.MakeSound();
            ((IAnimalInfo)elephant1).PrintInfo();
        }
    }
}
```

Banking System

```
using System;
namespace Termfinal
{
    abstract class Account
    {
        public string AccountNumber { get; set; }
        public string AccountHolderName { get; set; }
        public double Balance { get; set; }
        public abstract void Withdraw(double money);
        public abstract void Deposit(double money);
        protected void MakeDeposit(double money)
        {
            Balance += money;
            Console.WriteLine($"Sucessfully Deposited: {money}\nNew Balance : {Balance}");
        }
        protected void MakeWithdrawal(double money)
        {
            Balance -= money;
            Console.WriteLine($"Successful withdrawal.\nAmount: {money}, Remaining Balance : {Balance}");
        }
    }
    class SavingsAccount : Account
    {
        public override void Deposit(double money)
        {
            MakeDeposit(money);
        }

        public override void Withdraw(double money)
        {
            if (Balance < money)
            {
                Console.WriteLine($"Withdrawal cancelled! Insufficient banance : {Balance}.\n" +
                                $"5% of the remaining Balance : 0.05*{Balance} = {0.05 * Balance} has been ded
                                $"new Balance : {Balance - Balance*0.05}");
                Balance -= (Balance*0.05);
            }
            else
            {
                MakeWithdrawal(money);
            }

        }
    }
    class CheckingAccount : Account
    {
        public override void Deposit(double money)
        {
            MakeDeposit(money);
        }

        public override void Withdraw(double money)
        {
            if (Balance < money)
            {
                Console.WriteLine("Withdrawal cancelled! Insufficient banance.");
            }
            else
            {
                MakeWithdrawal(money);
            }
        }
    }
    class BusinessAccount : Account
    {
        public override void Deposit(double money)
        {
            MakeDeposit(money);
        }

        public override void Withdraw(double money)
        {
            if (Balance - money < -1000)
            {
                Console.WriteLine($"Withdrawal cancelled! Your Balance :{Balance}");
            }
            else
            {
                MakeWithdrawal(money);
            }
        }
    }
    class Program
    {
        public static void Main()
        {
            SavingsAccount savingsAccount = new SavingsAccount()
            {
                AccountNumber = "A123",
                AccountHolderName = "X1",
                Balance = 12345
            };
            CheckingAccount checkingAccount = new CheckingAccount()
            {
                AccountNumber = "B123",
                AccountHolderName = "X2",
                Balance = 14523
            };
            BusinessAccount businessAccount=new BusinessAccount{
                AccountNumber = "C123",
                AccountHolderName = "X3",
                Balance = 92345
            };
            savingsAccount.Deposit(100);
            businessAccount.Withdraw(10000);
            checkingAccount.Withdraw(1233);
        }
    }
}
```

CustomCollection

```
using System;

namespace Termfinal
{
    class CustomCollection<T>
    {
        private T[] customCollection;
        private int index = 0;

        public CustomCollection(int capacity)
        {
            this.Capacity = capacity;
            customCollection = new T[capacity];
        }

        public T this[int index]
        {
            get
            {
                if (index < 0 || index >= this.index)
                {
                    throw new IndexOutOfRangeException();
                }
                return customCollection[index];
            }
            set
            {
                if (index < 0 || index >= this.index)
                {
                    throw new IndexOutOfRangeException();
                }
                customCollection[index] = value;
            }
        }

        public int Capacity { get; }

        public int Count => index;

        public void Add(T value)
        {
            if (index == customCollection.Length)
            {
                throw new InvalidOperationException("Collection is at maximum capacity.");
            }
            customCollection[index] = value;
            index++;
        }

        public void RemoveAt(int x)
        {
            if (x < 0 || x >= index)
            {
                throw new IndexOutOfRangeException();
            }

            for (int i = x; i < index - 1; i++)
            {
                customCollection[i] = customCollection[i + 1];
            }
            index--;
        }

        public void Clear()
        {
            Array.Clear(customCollection, 0, customCollection.Length);
            index = 0;
        }

        public override string ToString()
        {
            return string.Join(" ", customCollection[..index]);
        }
    }

    class Program
    {
        public static void Main()
        {
            CustomCollection<int> customCollection = new CustomCollection<int>(10);

            for (int i = 0; i < 10; i++)
            {
                customCollection.Add(i);
            }

            customCollection.RemoveAt(4);
            Console.WriteLine(customCollection.ToString());
        }
    }
}
```

University Management System

```
using System;
using System.Data;
using System.Data.Common;
using System.Reflection;
namespace Termfinal
{
    abstract class Person
    {
        public string Name { get; set; }
        public int Id { get; set; }
        public int Age { get; set; }
    }
    class Student : Person
    {
        public int StudentId { get; set; }
        public string Department { get; set; }
        public double CGPA { get; set; }
    }
    class Professor : Person
    {
        public string EmployeeId { get; set; }
        public string Department { get; set; }
        public int Salary { get; set; }
    }
    class Course
    {
        public string CourseId { get; set; }
        public string Title { get; set; }
        public string Department { get; set; }
        public Professor ProfessoR { get; set; }
    }
    class University
    {
        private List<Course> courses = new List<Course>();
        private List<Student> students = new List<Student>();
        private List<Professor> professors = new List<Professor>();
        public string Name { get; set; }
        public University(string name)
        {
            this.Name = name;
        }
        public void Add(params Student[] students) => this.students.AddRange(students);
        public void Add(params Course[] courses) => this.courses.AddRange(courses);
        public void Add(params Professor[] professors) => this.professors.AddRange(professors);
        private void DoRemove<T>(List<T> list, params T[] rItems)
        {
            foreach (var item in rItems)
            {
                list.Remove(item);
            }
        }
        public void Remove(params Course[] courses) => DoRemove<Course>(this.courses, courses);
        public void Remove(params Student[] students) => DoRemove<Student>(this.students, students);
        public void Remove(params Professor[] professors) => DoRemove<Professor>(this.professors, professors);
        public delegate double AverageGPACalculator(List<Student> students);
        public double CalculateAverageGPA(AverageGPACalculator calculator)
        {
            return calculator(students);
        }
        public IEnumerable<Student> SortByCGPA()
        {
            return students.OrderByDescending(Student => Student.CGPA);
        }
    }

    class Program
    {
        static void Main()
        {
            University myUniversity = new University("My University");

            // Adding students
            Student student1 = new Student { Name = "Alice", Age = 20, StudentId = 1, Department = "Computer S
            Student student2 = new Student { Name = "Bob", Age = 21, StudentId = 2, Department = "Engineering"
            myUniversity.Add(student1, student2);

            // Adding professors
            Professor prof1 = new Professor { Name = "Dr. Smith", Age = 40, EmployeeId = "EMP001", Department 
            Professor prof2 = new Professor { Name = "Dr. Johnson", Age = 45, EmployeeId = "EMP002", Departmen
            myUniversity.Add(prof1, prof2);

            // Adding courses
            Course course1 = new Course { CourseId = "C001", Title = "Computer Networks", Department = "Comput
            Course course2 = new Course { CourseId = "C002", Title = "Mechanical Engineering", Department = "E
            myUniversity.Remove(course1, course2); // Removing initially added courses, as no link to universi
            myUniversity.Add(course1, course2);

            // Delegate usage: Calculating average GPA of students
            double averageGPA = myUniversity.CalculateAverageGPA(students => students.Average(s => s.CGPA));
            Console.WriteLine($"Average GPA of students: {averageGPA}");

            // LINQ operation: Sorting students by CGPA
            var sortedStudents = myUniversity.SortByCGPA();
            Console.WriteLine("Students sorted by CGPA:");
            foreach (var student in sortedStudents)
            {
                Console.WriteLine($"{student.Name} - CGPA: {student.CGPA}");
            }
        }
    }

}
```

Extension Class

```
using System;
namespace Termfinal
{
     public static class StringExtension{
        public static string Say(this string s)=>s;
     }
    class Program
    {
        enum Days { Sun, Mon = 3, Tue, Wed, Thu, Fri, Sat }
        static void Main(string[] args)
        {
            string s="hello";
            Console.Write(s.Say());
        }
    }
}
```