Module 2

What to learn

Understanding Classes
Classes Demo
Properties
Constructor
Static Methods
Extend a simple C# console app using class library

Practice Exercise

Practice 1
Do the hands C# video and practical given on https://docs.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-console-part-2?view=vs-2019
Practice 2
https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/intro-to-csharp/introduction-to-classes
Practice 3
https://docs.microsoft.com/en-us/dotnet/csharp/properties
Practice 4
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-constructors
Practice 5
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/static
Practice 6

Understanding Classes

Define a Class:
Write a simple class named Person with fields Name and Age.

Create an Object:
Create an object of the Person class in the Main method and assign values to its fields.

Instance vs. Class:
What is the difference between an instance of a class and the class itself? Explain with an example.

Access Modifiers:
Write a program to demonstrate public and private access modifiers in a class.

Class Usage:
Create a Car class with Brand, Model, and Price as fields. Use this class to print details of a car.

Practice 7
Field Initialization:
Write a Product class with fields Name, Category, and Price. Initialize them in the Main method

and print the details.

Encapsulation:
Modify the Product class to make its fields private and create methods to set and get values.

Array of Objects:
Create an array of Product objects, initialize 3 products, and print their details.

Methods in Classes:
Add a method CalculateDiscount to the Product class that calculates a discount based on price.

Class Relationship:
Create a class Order that contains a list of Product objects. Use the relationship to display details of all products in an order.

Practice 8

Properties

Auto-Implemented Properties:
Create a class Employee with Name, ID, and Salary as auto-implemented properties.

Custom Properties:
Add a custom property AnnualSalary in the Employee class that calculates and returns Salary * 12.

Read-Only Property:
Add a read-only property Department in the Employee class initialized in the constructor.

Validation with Properties:
Add validation in the Salary property to ensure it cannot be negative.

Default Value in Properties:
Use a property in a class that has a default value for Department.

Practice 9

Constructor

Parameterized Constructor:
Write a class Student with fields Name and Age. Use a parameterized constructor to initialize these fields.

Overloading Constructors:
Add a second constructor to the Student class that initializes only the Name field, setting a default value for Age.

Default Constructor:
Create a default constructor in the Student class that initializes default values for all fields.

Constructor Chaining:
Demonstrate constructor chaining in the Student class by calling one constructor from another.

Real-Life Scenario:
Write a class Book that requires Title and Author in its constructor and prints these details in a method.

Practice 10

Utility Class:
Create a static class MathUtilities with a static method Add that takes two numbers and returns their sum.

Static Counter:
Write a class with a static field to keep track of the number of objects created from the class.

Calling Static Methods:
Create a static method Greet in a class Helper and call it without creating an object of the class.

Non-Static with Static:
Create a non-static method that calls a static method from within the same class.

Static vs. Instance Methods:
Demonstrate the difference between static and instance methods using a simple class.
Practice 11

Mini-Project: Extend a Console App with a Class Library

Scenario:

You are tasked with building a **Student Management System** that uses a **class library** to handle business logic.

Steps:

Create a Class Library:

- Create a new project for a Class Library named BusinessLogic.
- Add a class StudentManager with the following methods:
  - AddStudent: Accepts Name and Age and adds a student to a list.
  - GetStudentDetails: Returns details of all students.
  - CalculateAverageAge: Calculates the average age of students.

Main Console Application:

- Create a Console App that references the BusinessLogic class library.
- In the Main method, use StudentManager to:
  - Add at least three students.
  - Display all student details.
  - Calculate and display the average age.

```
Student Management System
-------------------------
1. Add Student
2. View All Students
3. Calculate Average Age
4. Exit
-------------------------
Enter your choice: 1
Enter Name: John
Enter Age: 20

Enter your choice: 2
ID: 1, Name: John, Age: 20
```

```
Enter your choice: 3
Average Age: 20
```

Assignment Exercise

Assignment 1 (65cdfdab7ba36e06448762ce)
Create a reference type called Person. Populate the Person class with the following properties to store the following information: - First name - Last name - Email address - Date of birth Add constructors that accept the following parameter lists: - All four parameters - First, Last, Email - First, Last, Date of birth Add read-only properties that return the following computed information: - Adult – whether or not the person is over 18 - Sun sign – the traditional western sun sign of this person - Chinese sign – the chinese astrological sign (animal) of this person - Birthday – whether or not today is the person's birthday - Screen name – a default screen name that you might see being offered to a first time user of AOL or Yahoo (e.g. John Doe born on Feburary 25th, 1980 might be jdoe225 or johndoe022580) Access these things from Console Application in the Main Function. Accept this data for 5 person and display the same. Means create an object Array of 5 size and accept these details and display these details in tabular format.
Assignment 2

Assignment: Employee Payroll Management System

Objective:

Build a **C# Console Application** for an **Employee Payroll Management System**. Use a **Class Library** for the business logic. The system will allow users to manage employees, calculate payroll details, and demonstrate key concepts such as classes, properties, constructors, and static methods.

Requirements:

1. Create a Class Library (BusinessLogic):

Create a **Class Library** project named PayrollLibrary with the following components:

Class: Employee

- **Fields:** EmployeeID, Name, Department, BasicSalary, HRA, DA, Bonus.
- **Properties:**
    - Auto-implemented properties for EmployeeID, Name, and Department.
    - Validated properties for BasicSalary, HRA, DA, and Bonus to ensure they are non-negative.
- **Constructor:**
    - Parameterized constructor to initialize EmployeeID, Name, and Department.
    - Default constructor to set default values.

Class: PayrollManager

- **Static Method:** CalculateNetSalary(Employee employee)
    - Business Logic: NetSalary=BasicSalary+HRA+DA+Bonus\text{NetSalary} = \text{BasicSalary} + \text{HRA} + \text{DA} + \text{Bonus}NetSalary=BasicSalary+HRA+DA+Bonus

Instance Method: AddEmployee(Employee employee)

Adds an employee to a list.

Instance Method: DisplayEmployees()

Displays all employee details.

Main Console Application:

Create a **Console Application** project named EmployeePayrollApp. Add a reference to the PayrollLibrary class library.

Features in the Console App:

Main Menu

1. Add Employee
2. View All Employees
3. Calculate and Display Net Salary
4. Exit

Add Employee:

Accept EmployeeID, Name, Department, and BasicSalary from the user.
Automatically calculate HRA (20% of Basic Salary), DA (10% of Basic Salary), and Bonus (5% of Basic Salary).
Use the PayrollManager.AddEmployee method to add the employee to the system.

View All Employees:

Display a list of all employees using the PayrollManager.DisplayEmployees method.

Calculate and Display Net Salary:

Select an employee by ID.
Use the PayrollManager.CalculateNetSalary method to compute and display the net salary.

Deliverables:

A complete **C# solution** with:

PayrollLibrary Class Library.
EmployeePayrollApp Console Application.

Code should include:

Proper use of **classes**, **properties**, **constructors**, and **static methods**.
Business logic implemented in the class library for modularity and reusability.

Expected Learning Outcomes:

Develop an understanding of integrating class libraries in a console application.
Apply object-oriented programming principles to solve intermediate business problems.
Practice modular design and reusable components.

Online Reference

https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/intro-to-csharp/introduction-to-class https://docs.microsoft.com/en-us/dotnet/csharp/properties

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/usin

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/static

.NET Core Web API

WEB API (old)

Authentication And Authorization (WEBAPI)(old)

FullStackDevelopment_WIth_Dotnet_AND_Angular