Dotnet Batch - CSharp - Handout - Day 1

Notepad Content:

Microsoft Technologies Training

Programming -> C#
Database Layer -> MS SQL Server
Data Access Layer -> ADO.NET, Entity Framework Core
Business Layer -> C# Code
Service Layer -> ASP.NET Core Web API, Microservices

Presentation Layer -> ASP.NET Core MVC
Angular with Typescript

Cloud -> Microsoft Azure
Tools -> Git, DevOps

12 weeks
10 AM to 1.45 PM

10 AM to 11.30 AM -> Session 1

11.50 to 1.15 PM -> Session 2

1.15 to 1.45 -> Practice

2.30 pm to 5.30pm -> Assignments

Software Installation
-> Visual Studio 2022 Community Edition => DONE
-> Postman -> https://www.postman.com/downloads/
-> Nodejs -> https://nodejs.org/en

Command Prompt -> To check Dotnet version installed
dotnet v

Node.js Command Prompt -> To check node version installed
node -v

For Additional Learning Resources
-> Springboard - **Login using personal or college email ID only**
https://infyspringboard.onwingspan.com/web/en/login

-> Hello World Program

-> Classes, Objects and Methods

-> Class:
    Attributes, Fields, Instance members, Member variables
    Methods, Member methods, functions, Instance methods

```
Product p = new Product();
Reference = Object

Product p1 = new Product();
p1.Price = 100;
Product p2;
p2 = p1;
p1 = null;
```

```
Product p1 = new Product();
p1.Name = "A";
Product p2 = new Product();
p2.Name = "B";
```

Session Code:

Class Product.cs:

```
namespace A2ZSalesBusinessLayer
{
   public class Product
   {
      public int productId;
      public string productName;
      public string productDescription;
      public double price;
      public string category;
      public int quantityAvailable;

      public string DisplayDetails()
      {
         string details = "";
              details = productId + " " + productName + " " +
productDescription;
         return details;
      }

      public void UpdateProductStock(int addOnQuantity)
      {
            quantityAvailable = quantityAvailable + addOnQuantity;
        }
    }
}
```

Class Program.cs:

```
using A2ZSalesBusinessLayer;

namespace A2ZSalesConsoleApp
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Product productOne = new Product();

            //Set the data
            productOne.productId = 1;
            productOne.productName = "Mobile Phone";
            productOne.productDescription = "Smart phone for use";
            productOne.price = 10000;
            productOne.category = "Electronics";
            productOne.quantityAvailable = 10;

            //Get the data
            Console.WriteLine("Product Details : ");
            Console.WriteLine("Id : " + productOne.productId);
                                Console.WriteLine("Name   :   "   +
productOne.productName);
                            Console.WriteLine("Description   :   "   +
productOne.productDescription);
```

```
            Console.WriteLine("Price : " + productOne.price);
            Console.WriteLine("Category : " + productOne.category);
            Console.WriteLine("Quantity : " + productOne.quantityAvailable);

            //Invoke the method
            string details = productOne.DisplayDetails();
            Console.WriteLine("Details : " + details);

            productOne.UpdateProductStock(5);

            Console.WriteLine("New Quantity : " + productOne.quantityAvailable);
        }
    }
}
```

Assignments:

Assignment 1:

Create a simple C# application **GetWellApp** to represent a healthcare management system. The program should include the following classes:

1. Class: Doctor

- Fields:
  - `DoctorID` (int)
  - `Name` (string)
  - `Specialization` (string)

    - `ContactNumber` (long)
- Methods:
    - `DisplayDetails()` - Print the doctor's details.
    - `UpdateContactNumber(long newContactNumber)` - Update the doctor’s contact details.

2. Class: Patient

- Fields:
    - `PatientID` (int)
    - `Name` (string)
    - `Age` (int)
    - `Disease` (string)
- Methods:
    - `DisplayDetails()` - Print the patient's details.

3. Task Instructions:

1. Create 2 separate instances of the `Doctor` and `Patient` classes each in the `Main` method.
2. Initialize the fields using object initialization or by assigning values directly.
3. Call the `DisplayDetails()` method for both the doctor and the patient to display their details.
4. Call the `UpdateContactNumber()` method for the doctor and update the contact number of the second doctor.
5. Call the `DisplayDetails()` method for the second doctor to display the details.

4. Expected Output:

Example of how the program output might look:

Doctor Details:

```
ID: 101
Name: Dr. John Smith
Specialization: Cardiologist
Contact Number: 9876543210

Patient Details:
ID: 201
Name: Jane Doe
Age: 45
Disease: Hypertension

Doctor details after update:
Doctor Details:
ID: 101
Name: Dr. John Smith
Specialization: Cardiologist
Contact Number: 9999999999
```

Assignment 2:

Create a simple C# application **WeBank** to represent a banking system. The program should include the following classes:

1. Class: BankAccount

- Fields:
  - `AccountNumber` (int)
  - `AccountHolderName` (string)
  - `Balance` (decimal)
- Methods:
  - `Deposit(decimal amount)` - Add the amount to the balance and display the updated balance.
  - `Withdraw(decimal amount)` - Subtract the amount from the balance if sufficient funds are available. If not, display an error message.
  - `DisplayAccountDetails()` - Print the account details.

2. Class: Transaction

- Fields:
  - `TransactionID` (int)
  - `AccountNumber` (int)
  - `TransactionType` (string) // "Deposit" or "Withdrawal"
  - `Amount` (decimal)
- Methods:
  - `DisplayTransactionDetails()` - Print the transaction details.

3. Task Instructions:

1. Create an instance of the `BankAccount` class in the `Main` method.
2. Initialize the fields with sample data.
3. Perform the following actions:
   - Deposit an amount into the account and display the updated balance.

    - Withdraw an amount from the account, ensuring that there are sufficient funds.
4. Create an instance of the `Transaction` class to record each transaction.
5. Display the account details and the details of the transactions performed.

4. Expected Output:

Example of how the program output might look:

```
Account Details:

Account Number: 123456

Account Holder: John Doe

Balance: $5000.00


Deposit successful! New Balance: $6000.00

Withdrawal successful! New Balance: $5500.00


Transaction Details:

Transaction ID: 1

Account Number: 123456

Transaction Type: Deposit

Amount: $1000.00
```

Transaction ID: 2

Account Number: 123456

Transaction Type: Withdrawal

Amount: $500.00