**Roll No: 412084**

**Prn No: 2019033800126574**

**Name: YAGNIK MOJIDRA**

**“Assignment -4”**

**Github::** <https://github.com/YagnikMojidra/Dot-Net-workspace>

**Question 1:: Indexer and classes properties**

**Code::**

using System;

//Properties with backing fields

//print Time period

class TimePeriod

{

private double seconds;

public double Hours

{

get { return seconds / 3600; }

set

{

if (value < 0 || value > 24)

throw new ArgumentOutOfRangeException(

$"{nameof(value)} must be between 0 and 24.");

seconds = value \* 3600;

}

}

}

//Expression body definitions

// print name

public class Person

{

private string firstName;

private string lastName;

public Person(string first, string last)

{

firstName = first;

lastName = last;

}

public string Name => $"{firstName} {lastName}";

}

//Expression body definitions

public class SaleItem

{

string \_name;

decimal \_cost;

public SaleItem(string name, decimal cost)

{

\_name = name;

\_cost = cost;

}

public string Name

{

get => \_name;

set => \_name = value;

}

public decimal Price

{

get => \_cost;

set => \_cost = value;

}

}

//Auto-implemented properties

public class SaleItem2

{

public string Name2

{ get; set; }

public decimal Price2

{ get; set; }

}

//Using indexers

public class TempRecord

{

// Array of temperature values

float[] temps = new float[10]

{

56.2F, 56.7F, 56.5F, 56.9F, 58.8F,

61.3F, 65.9F, 62.1F, 59.2F, 57.5F

};

// To enable client code to validate input

// when accessing your indexer.

public int Length => temps.Length;

// Indexer declaration.

// If index is out of range, the temps array will throw the exception.

public float this[int index]

{

get => temps[index];

set => temps[index] = value;

}

}

class Program

{

static void Main()

{

TimePeriod t = new TimePeriod();

t.Hours = 24;

Console.WriteLine($"Time in hours is: {t.Hours}");

var person = new Person("Yagnik", "Mojidra");

Console.WriteLine(person.Name);

var item = new SaleItem("Shoes", 19.95m);

Console.WriteLine($"{item.Name}: sells for {item.Price:C2}");

var item2 = new SaleItem2 { Name2 = "Shoes", Price2 = 19.95m };

Console.WriteLine($"{item2.Name2}: sells for {item2.Price2:C2}");

var tempRecord = new TempRecord();

// Use the indexer's set accessor

tempRecord[3] = 58.3F;

tempRecord[5] = 60.1F;

tempRecord[8] = 62.1F;

// Use the indexer's get accessor

for (int i = 0; i < 10; i++)

{

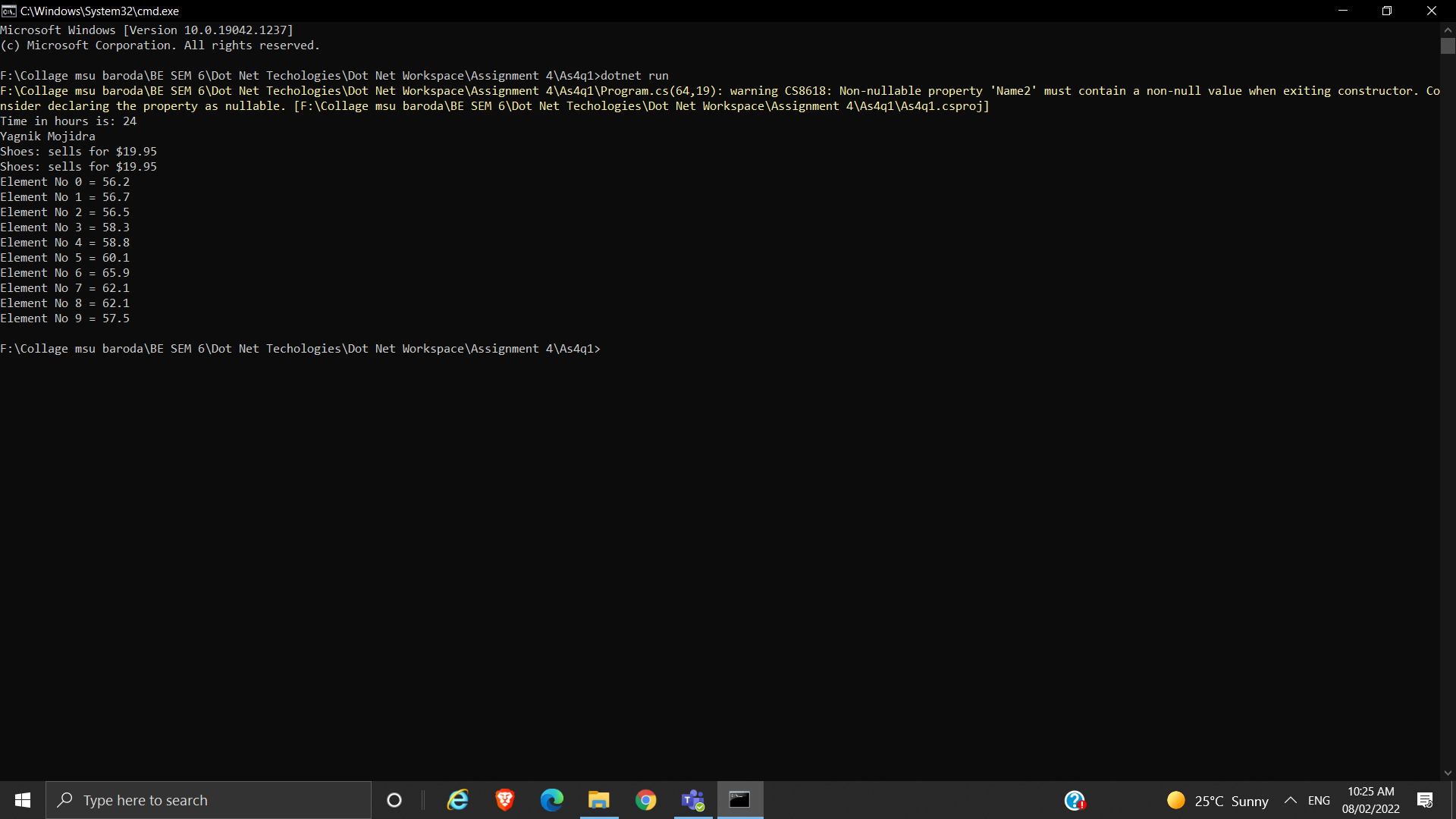
Console.WriteLine($"Element No {i} = {tempRecord[i]}");

}

}

}

**Output:**

****

**Question 2:: Employee class question**

**Code::**

using System;

// public DateOnly rdate { get; }

public class Employee

{

public string firstName { get; set; }

public string lastName { get; set; }

public double monthlySalary

{

get;

set;

}

public override string ToString()

{

return $"{firstName} {lastName} {monthlySalary}";

}

public double giveRaise(double msalary)

{

return msalary += msalary \* 0.1;

}

}

public class permanentEmployee : Employee

{

public double hra { get; set; }

public double da { get; set; }

public double pf { get; set; }

public override string ToString()

{

return $"{firstName} \t {lastName} \t {monthlySalary} \t {hra} \t {da} \t {pf}";

}

public double giveRaise(double msalary, double hra, double da, double pf)

{

return msalary =msalary+ msalary\*0.01 + hra + da + pf;

}

}

class TestEmployee

{

static void Main()

{

//for question1

var emp1 = new Employee { firstName = "Yagnik", lastName = "Mojidra", monthlySalary = 24000 };

var emp2 = new Employee { firstName = "Utsav", lastName = "Patel", monthlySalary = 34000 };

if (emp1.monthlySalary < 0 || emp2.monthlySalary < 0)

{

Console.WriteLine("Salary should be greater than zero.");

emp1.monthlySalary = 0;

emp2.monthlySalary = 0;

Console.WriteLine($"{emp1.firstName} {emp1.lastName} your yearly salary is : {emp1.monthlySalary} .");

Console.WriteLine($"{emp2.firstName} {emp2.lastName} your yearly salary is : {emp2.monthlySalary} .");

}

else

{

Console.WriteLine($"{emp1.firstName} {emp1.lastName} your yearly salary is : {emp1.monthlySalary \* 12} .");

Console.WriteLine($"{emp2.firstName} {emp2.lastName} your yearly salary is : {emp2.monthlySalary \* 12} .");

}

Console.WriteLine("Here we use ToString Method.");

Console.WriteLine("FirstName\tLastName\tMonthlySalary");

Console.WriteLine(emp1.ToString());

Console.WriteLine(emp2.ToString());

Console.WriteLine("Our employees get 10% increment in salary.");

emp1.giveRaise(emp1.monthlySalary);

emp2.giveRaise(emp2.monthlySalary);

Console.WriteLine($"{emp1.firstName} {emp1.lastName} your yearly salary with increment is : {emp1.monthlySalary \* 12} .");

Console.WriteLine($"{emp2.firstName} {emp2.lastName} your yearly salary with increment is : {emp2.monthlySalary \* 12} .");

//for question2

//here we use method overloading and hiding

var emp3 = new permanentEmployee { firstName = "Yagnik", lastName = "Mojidra", monthlySalary = 24000, hra = 24000 \* 0.08, da = 24000 \* .06, pf = 24000 \* 0.02 };

DateOnly jdate = new DateOnly(2022, 2, 5);

Console.WriteLine("Here we use ToString Method.");

Console.WriteLine("FirstName\tLastName\tMonthlySalary\tHRA\tDA\tPF");

Console.WriteLine(emp3.ToString());

Console.WriteLine("This employee Joing date is " + jdate.ToString());

DateOnly rdate = new DateOnly(2026, 2, 5);

Console.WriteLine("This employee Expected Retirement date is " + rdate.ToString());

// emp3.giveRaise(emp3.monthlySalary, emp3.hra, emp3.da, emp3.pf);

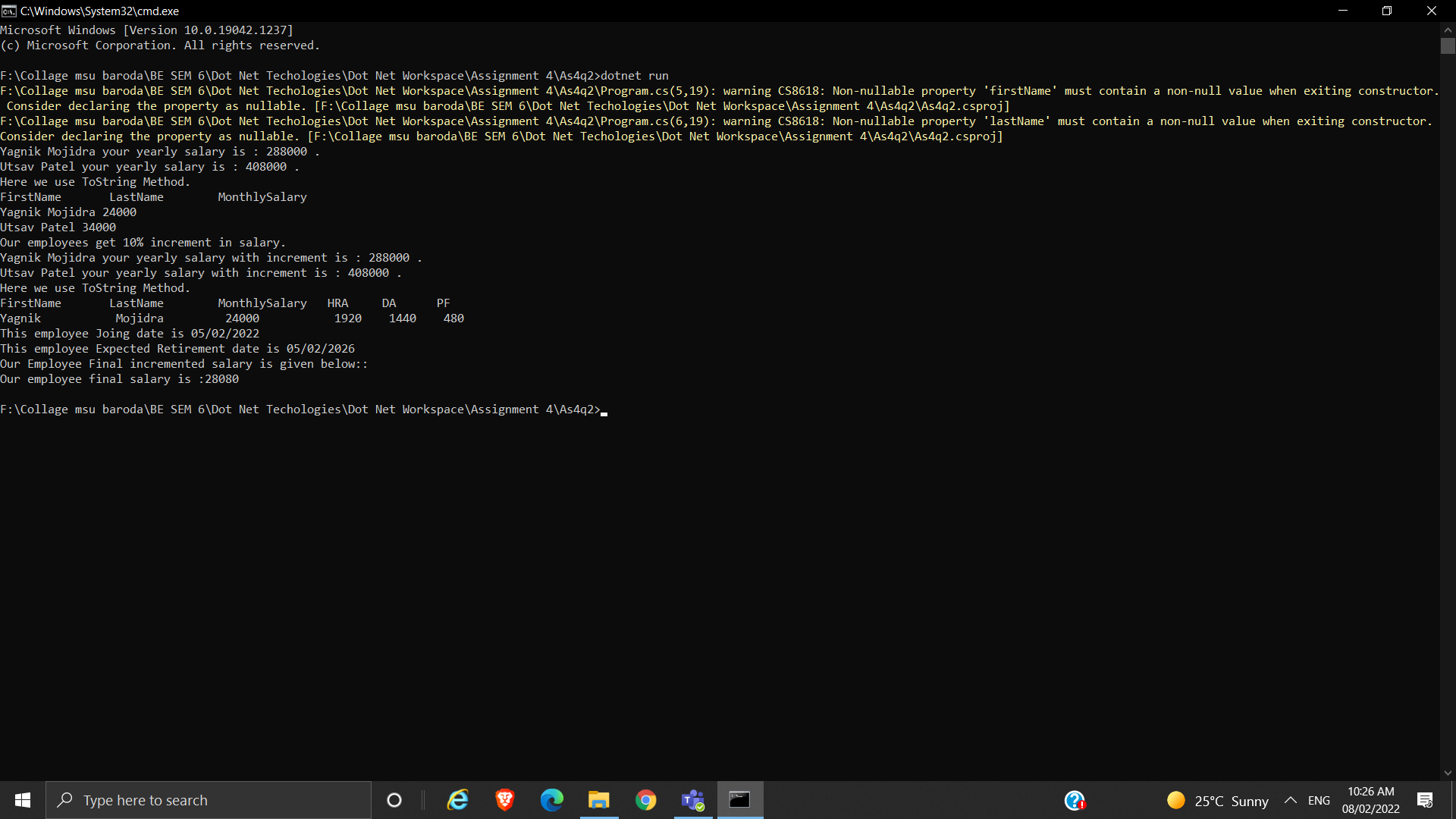
Console.WriteLine("Our Employee Final incremented salary is given below:: ");

Console.WriteLine("Our employee final salary is :" + emp3.giveRaise(emp3.monthlySalary, emp3.hra, emp3.da, emp3.pf));

}

}

**Output:**

****

**Question 3:: method question**

**Code::**

using System;

using System.Reflection;

public class SimpleClassExample

{

public static void Main()

{

Type t = typeof(SimpleClassExample);

BindingFlags flags = BindingFlags.Instance | BindingFlags.Static | BindingFlags.Public |

BindingFlags.NonPublic | BindingFlags.FlattenHierarchy;

MemberInfo[] members = t.GetMembers(flags);

Console.WriteLine($"Type {t.Name} has {members.Length} members: ");

foreach (var member in members)

{

string access = "";

string stat = "";

var method = member as MethodBase;

if (method != null)

{

if (method.IsPublic)

access = " Public";

else if (method.IsPrivate)

access = " Private";

else if (method.IsFamily)

access = " Protected";

else if (method.IsAssembly)

access = " Internal";

else if (method.IsFamilyOrAssembly)

access = " Protected Internal ";

if (method.IsStatic)

stat = " Static";

}

var output = $"{member.Name} ({member.MemberType}): {access}{stat}, Declared by {member.DeclaringType}";

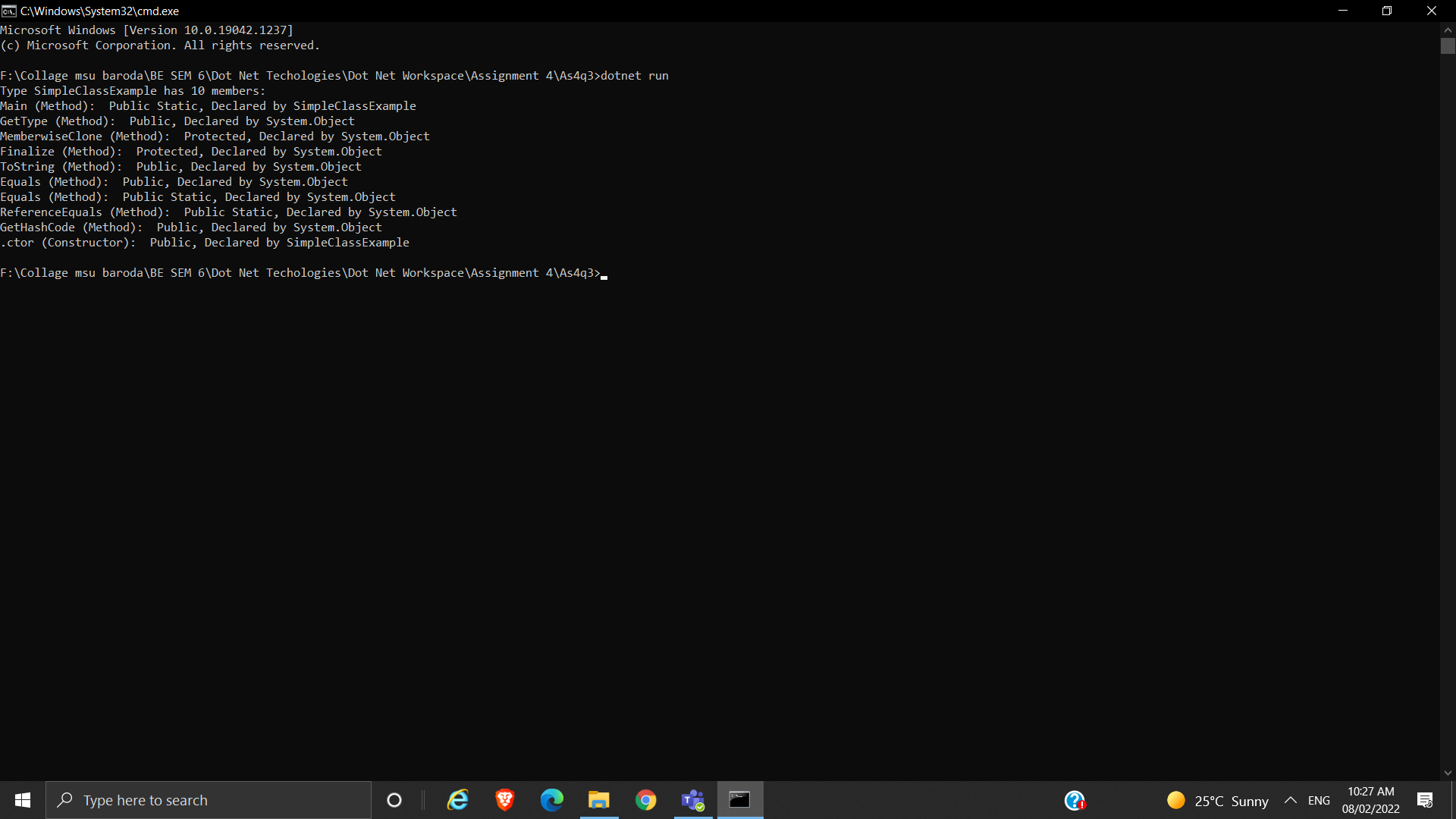
Console.WriteLine(output);

}

}

}

**Output:**

****