Lab - 4

1.

Code :-

```
namespace Program
{

    class MainClass
    {

        public static void Main(String[] argv)
        {

            EmployeeData.Employee employee1 = new EmployeeData.Employee("Smit", "Shah",
100000.00);
            EmployeeData.Employee employee2 = new EmployeeData.Employee("Harsh", "Patel",
250000.00);

            //details of employee 1 & employee 2
            Console.WriteLine("-----------Yearly Salary Of Employees----------- \n");
            Console.WriteLine($"-------First Employee-------  \nYearly Salary :
Rs.{employee1.MonthlySalary * 12}");
            Console.WriteLine($"-------Second Employee------- \nYearly Salary :
Rs.{employee2.MonthlySalary * 12}");


            //giving 10% raise to employee
            employee1.getRaise(10);
            employee2.getRaise(10);

            Console.WriteLine("\n-----------Yearly Salary Of Employees After Giving
10%Raise-----------\n");
            Console.WriteLine($"-------First Employee-------  \n\nYearly Salary :
{employee1.MonthlySalary * 12}");
            Console.WriteLine($"-------Second Employee------- \n\nYearly Salary :
{employee2.MonthlySalary * 12}");


            //permanent Employees Derived Class
            EmployeeData.PermanentEmployee permanentEmployee1 = new
EmployeeData.PermanentEmployee("Smit", "Shah", 100000.00, "14-02-2022", "20-12-2032");
            EmployeeData.PermanentEmployee permanentEmployee2 = new
EmployeeData.PermanentEmployee("Harsh", "Patel", 250000.00, "14-02-2022", "06-12-2042");
            Console.WriteLine("\n\n-----------For Permanent Employees-----------");
            Console.WriteLine("\n-------Details Of Permanent Employee 1-------\n");
            Console.WriteLine(permanentEmployee1);
            Console.WriteLine("\n-------Details Of Permanent Employee 2-------");
            Console.WriteLine(permanentEmployee2);
```

```
            //giving 20% raise
            permanentEmployee1.getRaise(20);
            permanentEmployee2.getRaise(20);


            Console.WriteLine("\n\n-----------After Giving 20% Raise-----------");
            Console.WriteLine("\n-------Details Of Permanent Employee 1-------\n");
            Console.WriteLine(permanentEmployee1);
            Console.WriteLine("\n-------Details Of Permanent Employee 2-------");
            Console.WriteLine(permanentEmployee2);



        }
    }
}
```

```
using System;

namespace EmployeeData
{

    class Employee
    {

        public string _firstName;
        public string _lastName;
        public double _monthlySalary;

        //creating getter and setter properties

        public string FirstName
        {

            get => _firstName;
            set => _firstName = value;

        }
        public string LastName
        {

            get => _lastName;
            set => _lastName = value;

        }

        public double MonthlySalary
        {
```

```
            get => _monthlySalary;

            set
            {

                if (value < 0)
                    value = 0.0;

                _monthlySalary = value;

            }
        }

        //constructor
        public Employee(string firstName, string lastName, double monthlySalary)
        {

            _firstName = firstName ?? throw new Exception();
            _lastName = lastName ?? throw new Exception();
            _monthlySalary = monthlySalary;


        }

        //creating overridable method giveRaise()...
        public virtual void getRaise(double raise)
        {

            _monthlySalary += _monthlySalary * (raise / 100);

        }

        //overriding ToString from object class

        public override string ToString()
        {
            return $"Employee Name : {_firstName} {_lastName}  \nMonthly Salary :
Rs.{_monthlySalary}";


        }


    }


    class PermanentEmployee : Employee
    {

        private double _hra;
        private double _da;
        private double _pf;
```

```
        string _joiningDate;
        string _retirementDate;

        public PermanentEmployee(string firstName, string lastName, double monthlySalary,
string joiningDate, string retirementDate) : base(firstName, lastName, monthlySalary)
        {
            _da = base._monthlySalary * 0.12;
            _hra = (_da + _monthlySalary) * 0.05;
            _pf = _da;
            _joiningDate = joiningDate;
            _retirementDate = retirementDate;

        }

        //setting only read only properties
        public double HRA
        {

            get => _hra;
        }
        public double DA
        {

            get => _da;
        }
        public double PF
        {

            get => _pf;
        }

        public override void getRaise(double raise)
        {
            base.getRaise(raise);
            _monthlySalary += _hra + _da;

        }

        public override string ToString()
        {
            return $"{base.ToString()}\nJoining date: {_joiningDate}\nRetirement
date:{_retirementDate}";
        }

    }
}
```

Output :-

```
-----------Yearly Salary Of Employees-----------

-------First Employee-------
Yearly Salary : Rs.1200000
-------Second Employee-------
Yearly Salary : Rs.3000000

-----------Yearly Salary Of Employees After Giving 10%Raise-----------

-------First Employee-------

Yearly Salary : 1320000
-------Second Employee-------

Yearly Salary : 3300000
```

```
 -----------For Permanent Employees-----------

 -------Details Of Permanent Employee 1-------

 Employee Name : Smit Shah
 Monthly Salary : Rs.100000
 Joining date: 14-02-2022
 Retirement date:20-12-2032

 -------Details Of Permanent Employee 2-------
 Employee Name : Harsh Patel
 Monthly Salary : Rs.250000
 Joining date: 14-02-2022
 Retirement date:06-12-2042


 -----------After Giving 20% Raise-----------

 -------Details Of Permanent Employee 1-------

 Employee Name : Smit Shah
 Monthly Salary : Rs.137600
 Joining date: 14-02-2022
 Retirement date:20-12-2032

 -------Details Of Permanent Employee 2-------
 Employee Name : Harsh Patel
 Monthly Salary : Rs.344000
 Joining date: 14-02-2022
 Retirement date:06-12-2042
```

2.

Code :-

```
using System;
using Indexers;
namespace MainProgram
{
    class Program
    {
        public static void Main(string[] args)
        {
            //example-1
            var tempRecord = new TempRecord();

            tempRecord[3] = 58.3F;
            tempRecord[5] = 60.1F;

            for (int i = 0; i < 10; i++)
            {
                Console.WriteLine($"Element #{i} = {tempRecord[i]}");
            }
            //example-2
            var week = new DayCollection();
            Console.WriteLine(week["Fri"]);

            try
            {
                Console.WriteLine(week["Made-up day"]);
            }
            catch (ArgumentOutOfRangeException e)
            {
                Console.WriteLine($"Not supported input: {e.Message}");
            }
            //example-3
            var week2 = new DayOfWeekCollection();
            Console.WriteLine(week2[DayOfWeek.Friday]);

            try
            {
                Console.WriteLine(week2[(DayOfWeek)43]);
            }
            catch (ArgumentOutOfRangeException e)
            {
                Console.WriteLine($"Not supported input: {e.Message}");
            }

        }
    }
}
```

```
using System;
```

```
using Day = System.DayOfWeek;
namespace Indexers
{
    public class TempRecord
    {
        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
     };


        public int Length => temps.Length;


        public float this[int index]
        {
            get => temps[index];
            set => temps[index] = value;
        }
    }


    class DayCollection
    {
        string[] days = { "Sun", "Mon", "Tues", "Wed", "Thurs", "Fri", "Sat" };

        public int this[string day] => FindDayIndex(day);

        private int FindDayIndex(string day)
        {
            for (int j = 0; j < days.Length; j++)
            {
                if (days[j] == day)
                {
                    return j;
                }
            }

            throw new ArgumentOutOfRangeException(
                nameof(day),
                $"Day {day} is not supported.\nDay input must be in the form \"Sun\",
\"Mon\", etc");
        }
    }

    class DayOfWeekCollection
    {
        Day[] days =
        {
        Day.Sunday, Day.Monday, Day.Tuesday, Day.Wednesday,
        Day.Thursday, Day.Friday, Day.Saturday
    };
```

```
        // Indexer with only a get accessor with the expression-bodied definition:
        public int this[Day day] => FindDayIndex(day);

        private int FindDayIndex(Day day)
        {
            for (int j = 0; j < days.Length; j++)
            {
                if (days[j] == day)
                {
                    return j;
                }
            }
            throw new ArgumentOutOfRangeException(
                nameof(day),
                $"Day {day} is not supported.\nDay input must be a defined
System.DayOfWeek value.");
        }
    }

}
```

Output:-

```
Element #0 = 56.2
Element #1 = 56.7
Element #2 = 56.5
Element #3 = 58.3
Element #4 = 58.8
Element #5 = 60.1
Element #6 = 65.9
Element #7 = 62.1
Element #8 = 59.2
Element #9 = 57.5
5
Not supported input: Day Made-up day is not supported.
Day input must be in the form "Sun", "Mon", etc (Parameter 'day')
5
Not supported input: Day 43 is not supported.
Day input must be a defined System.DayOfWeek value. (Parameter 'day')
```

3.

Code:-

```
using System;

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;
        }
    }

}

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;
    }
}

public class SaleItems
{
    public string Name
    { get; set; }

    public decimal Price
    { get; set; }
}



class Program
{
```

```
    static void Main()
    {
        TimePeriod t = new TimePeriod();
        t.Hours = 24;
        Console.WriteLine($"Time in hours : {t.Hours}");

        var item = new SaleItem("Shoes", 19.95m);
        Console.WriteLine($"{item.Name}: sells for {item.Price:C2}");

        var newItem = new SaleItems { Name = "Shoes", Price = 19.95m };
        Console.WriteLine($"{newItem.Name}: sells for {newItem.Price:C2}");
    }
}
```

Output :-

```
Time in hours : 24
Shoes: sells for £19.95
Shoes: sells for £19.95
```

4.

Code :-

```
using System;
using System.Reflection;


public class SampleClass
{

}
public class SimpleClassExample
{
    public static void Main()
    {
        Type t = typeof(SampleClass);
        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 :-

```
Type SampleClass has 9 members:
GetType (Method):  Public, Declared by System.Object
MemberwiseClone (Method):  Protected, Declared by System.Object
Finalize (Method):  Protected, Declared by System.Object
ToString (Method):  Public, Declared by System.Object
Equals (Method):  Public, Declared by System.Object
Equals (Method):  Public Static, Declared by System.Object
ReferenceEquals (Method):  Public Static, Declared by System.Object
GetHashCode (Method):  Public, Declared by System.Object
.ctor (Constructor):  Public, Declared by SampleClass
```