**.NET Assignment - 4**

Github link : <https://github.com/Aishwarya01-github/.NET-Assignments/tree/main/Lab%204>

Code :-

using System;

namespace Code\_1

{

class Program

{

static void Main(string[] args)

{

TimePeriod t = new TimePeriod();

// The property assignment causes the 'set' accessor to be called.

t.Hours = 24;

// Retrieving the property causes the 'get' accessor to be called.

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

var person = new Person("Aishwarya", "Mundley");

Console.WriteLine(person.Name);

var item = new SaleItem { Name = "Shoes", Price = 19.95m };

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

}

}

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 Person

{

private string \_firstName;

private string \_lastName;

public Person(string first, string last)

{

\_firstName = first;

\_lastName = last;

}

public string Name => $"{\_firstName} {\_lastName}";

}

public class SaleItem

{

public string Name

{ get; set; }

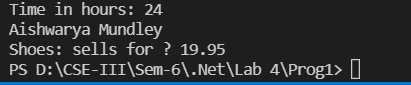
public decimal Price

{ get; set; }

}

}

Output :



Code 2:

using System;

using Day = System.DayOfWeek;

namespace Code\_2

{

class Program

{

static void Main(string[] args)

{

// Example 1

var tempRecord = new TempRecord();

// Use the indexer's set accessor

tempRecord[3] = 58.3F;

tempRecord[5] = 60.1F;

// Use the indexer's get accessor

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

{

Console.WriteLine($"Element #{i} = {tempRecord[i]}");

}

// Keep the console window open in debug mode.

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

// 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

}

}

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 DayCollection

{

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

// Indexer with only a get accessor with the expression-bodied definition:

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),

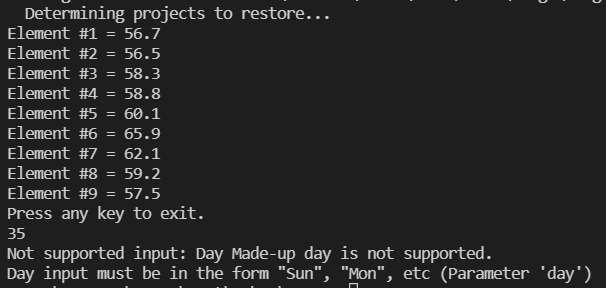
$"Day {day} is not supported.\nDay input must be in the form \"Sun\", \"Mon\", etc");

}

}

}

Output :



Code 3:

using System;

namespace Code\_3

{

class Program

{

static void Main(string[] args)

{

Employee e1 = new Employee("Aishwarya", "Mundley", 45000);

Employee e2 = new Employee("Aditya", "Mundley", 23000);

Console.WriteLine("Before Increament...");

Console.WriteLine(e1.ToString());

Console.WriteLine(e2.ToString());

Console.WriteLine("After Increament...");

e1.giveRaise(10.0);

e2.giveRaise(10.0);

Console.WriteLine(e1.ToString());

Console.WriteLine(e2.ToString());

Console.WriteLine("----------- Permanent Employee ---------");

PermanentEmployee pe1 = new PermanentEmployee("Aishwarya", "Mundley", 45000, 2000, 1000, 4500, "01-02-2022", "10-05-2022");

PermanentEmployee pe2 = new PermanentEmployee("Aditya", "Mundley", 23000, 2789, 500, 9500, "01-01-2022", "18-12-2022");

Console.WriteLine("Before Increament...");

Console.WriteLine(pe1);

Console.WriteLine(pe2);

pe1.giveRaise(10.0);

pe2.giveRaise(10.0);

Console.WriteLine("After Increament...");

Console.WriteLine(pe1.ToString());

Console.WriteLine(pe2.ToString());

}

}

public class Employee

{

private String \_firstName;

private String \_lastName;

private double \_monSalary;

public Employee(String first, String last, double sal)

{

\_firstName = first;

\_lastName = last;

\_monSalary = sal;

}

public String First

{

get => \_firstName;

set => \_firstName = value;

}

public String Last

{

get => \_lastName;

set => \_lastName = value;

}

public double MonSalary

{

get => \_monSalary;

set

{

if (value < 0.0)

{

\_monSalary = 0.0;

}

else

{

\_monSalary = value;

}

}

// 2nd way

// set => \_monSalary = value < 0.0 ? 0.0 : value;

}

public virtual void giveRaise(double inc)

{

\_monSalary = \_monSalary + (\_monSalary \* inc / 100);

}

public override string ToString()

{

return "Employee Details : " + \_firstName + " " + \_lastName + " Yearly Salary : " + (\_monSalary) \* 12;

}

}

public class PermanentEmployee : Employee

{

private double \_hra;

private double \_da;

private double \_pf;

private String \_joiningDate;

private String \_retirementDate;

public PermanentEmployee(String first, String last, double sal, double hra, double da, double pf, String joiningDate, String retirementDate) : base(first, last, sal)

{

this.\_hra = hra;

this.\_da = da;

this.\_pf = pf;

this.\_joiningDate = joiningDate;

this.\_retirementDate = retirementDate;

this.MonSalary = this.MonSalary + \_hra + \_da;

}

public double Hra

{

get => \_hra;

}

public double Da

{

get => \_da;

}

public double Pf

{

get => \_pf;

}

public String JoiningDate

{

get => \_joiningDate;

set => \_joiningDate = value;

}

public String RetirementDate

{

get => \_retirementDate;

set => \_retirementDate = value;

}

public override void giveRaise(double inc)

{

this.MonSalary = this.MonSalary + (this.MonSalary \* inc) / 100 + \_da + \_hra;

}

public override string ToString()

{

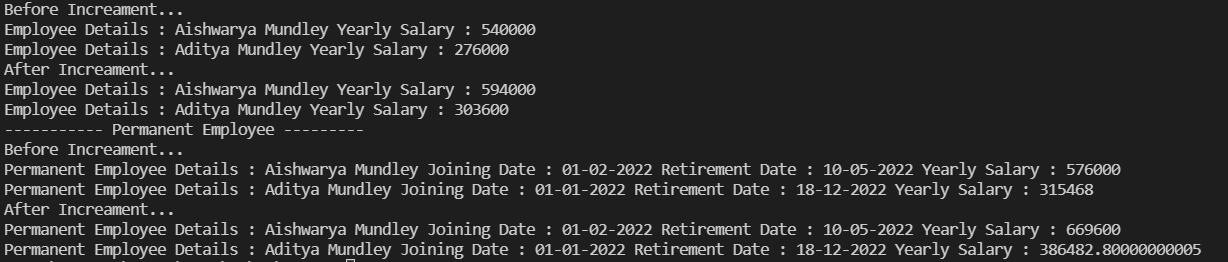
return "Permanent Employee Details : " + this.First + " " + this.Last + " Joining Date : " + \_joiningDate + " Retirement Date : " + \_retirementDate + " Yearly Salary : " + (this.MonSalary) \* 12;

}

}

}

Output :

SampleExample:

using System;

using System.Reflection;

namespace SimpleClassExample

{

class Program

{

static void Main(string[] args)

{

Type t = typeof(SimpleClass);

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);

}

}

}

public class SimpleClass

{

}

}

Output :

