1) Properties

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;

       }

   }

}

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{

   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 AutoSaleItem{

   public string? Name

   { get; set; }

   public decimal Price

   { get; set; }

}

class Test{

   static void Main(){

     Console.WriteLine("By: Mihir L. Agrawal\n");

      TimePeriod t = new TimePeriod();

      t.Hours = 12;

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

      Person p = new Person("Mihir","Agrawal");

      Console.WriteLine($"The Name of Person: {p.Name}\n");

      SaleItem s = new SaleItem("Book",99.0m);

      Console.WriteLine($"The {s.Name} costs : {s.Price}\n");

      //Auto implemented

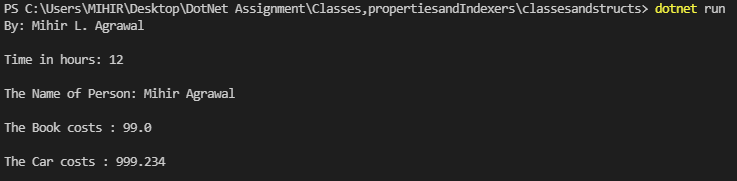
      AutoSaleItem asi = new AutoSaleItem{Name = "Car",Price = 999.234m};

      Console.WriteLine($"The {asi.Name} costs : {asi.Price}\n");

   }

}

Ouptut:



2) Indexers

using System;

namespace Indexers{

  class Collection<T>{

    private T[] arr = new T[100];

    public T this[int i]{

      get{return arr[i];}

      set{arr[i]=value;}

    }

  }

  class vector<T>{

    private T[] arr = new T[5];

    public int curr = 0,size\_ = 5;

    public T this[int i]{

      get{

      if(i>=size\_ || i < 0)

        throw new IndexOutOfRangeException("Out of bounds Accessing is not possible\n");

        return arr[i];

      }

      set{

        if(i>=size\_ || i < 0)

          throw new IndexOutOfRangeException("Out of bounds Accessing is not possible\n");

        arr[i] = value;

      }

    }

    public void push\_back(T value){

      if(curr==size\_)

          throw new IndexOutOfRangeException($"Max Size of array is {size\_}\n");

      arr[curr++] = value;

    }

  }

  class test{

    public static void Main(){

      Collection<double> salary = new Collection<double>();

      salary[0] = 1.04;

      salary[1] = 20.10;

      salary[2] = 40.23;

      for(int i = 0;i < 3;i++)

        Console.Write(salary[i]+" ");

      Console.Write("\n\n");

      vector<string> arr = new vector<string>();

      string[] lyrics = {"First things","First","I'm gonna" ,"say all the words","inside my head"};

      foreach(string s in lyrics){

        arr.push\_back(s);

      }

      for(int i = 0; i < arr.curr;i++){

        Console.Write(arr[i] + " ");

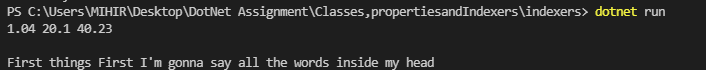
      }

    }

  }

}

Output:



3) Doc -1

using System.Reflection;

public class SimpleClass

{

    public static void Main()

    {

        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}";

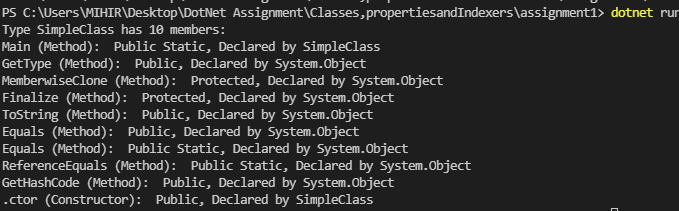
            Console.WriteLine(output);

        }

    }

}

Output:



4) Doc -2 answers

Employee Class

// create a class called Employee that includes three pieces of information as fields—a first name (type

// String), a last name (type String) and a monthly salary (type double). Your class should have a constructor that

// initializes the fields.

// Provide a getter setter properties for each field. If the monthly salary is not positive, set it to 0.0.

//Create an overridable method named giveRaise(). Override the ToSTring() method

// from object class appropriately here. Write a test application named EmployeeTest

// that demonstrates class Employee’s capabilities. Create two Employee objects and display the yearly salary for each

// Employee. Then give each Employee a 10% raise and display each Employee’s yearly salary again.

// Create a class Permanent employee that is derived from employee class and it includes extra fields HRA (hoursing rent allowance), DA (dearness allowance)

// and Provident Fund. Choose the field types appropriately and also create read only properites for each of these fields.

// Create  fields named Joining Date and Expected Retirement Date. Also create properties for both the fields.

// Ensure to create constructors and appropriate methods (atleast two). Override the giveRaise() method from parent class and

// ensure that now along with %raise in salary, the hra and da are added and the final salary is appropriately calculated.

// Override the ToString() method appropriately for a permanant employee object.

using System;

namespace  EmployeeSpace{

  class Employee{

    private string firstname,lastname;

    double salary;

    public string Firstname{

      get => firstname;

      set => firstname = String.Empty + value;

    }

    public string Lastname{

      get => lastname;

      set => lastname = String.Empty + value;

    }

    public double Salary{

      get => salary;

      set => salary = Math.Max(0.0,value);

    }

    public Employee(string firstname,string lastname,double salary){

      this.firstname = String.Empty + firstname;

      this.lastname = String.Empty + lastname;

      Salary = Math.Max(0.0,salary);

    }

    //Overriding Object method ToString

    override public string ToString(){

      return ("Name: " + Firstname + " " + Lastname + "\nSalary: " + Salary + "\n");

    }

    virtual public void giveRaise(double perc = 10){

      Salary \*= (1+(perc/100));

    }

  }

  class PermanentEmployee : Employee{

    static private double HRA = 100,DA = 100,PF = 100;

    private DateOnly joinDate,retireDate;

    public DateOnly JoinDate{

      get=>joinDate;

      set=>joinDate=value;

    }

    public DateOnly RetireDate{

      get=>retireDate;

      set=>retireDate=value;

    }

    public PermanentEmployee(string firstname,string lastname,double salary) : base(firstname,lastname,salary){

      //new is needed to create Dateonly objects because it non primitive

      Console.Write($"Enter the Year(YYYY) of Joining of {Firstname + " " + Lastname}:");

      int y = Convert.ToInt32(""+Console.ReadLine());

      Console.Write($"Enter the Month(MM) of Joining of {Firstname + " " + Lastname}:");

      int m = Convert.ToInt32(""+Console.ReadLine());

      Console.Write($"Enter the Date(DD) of Joining of {Firstname + " " + Lastname}:");

      int d = Convert.ToInt32(""+Console.ReadLine());

      joinDate = new DateOnly(y,m,d);

      Console.Write($"Enter the Year(YYYY) of Retiring of {Firstname + " " + Lastname}:");

      y = Convert.ToInt32(""+Console.ReadLine());

      Console.Write($"Enter the Month(MM) of Retiring of {Firstname + " " + Lastname}:");

      m = Convert.ToInt32(""+Console.ReadLine());

      Console.Write($"Enter the Date(DD) of Retiring of {Firstname + " " + Lastname}:");

      d = Convert.ToInt32(""+Console.ReadLine());

      retireDate = new DateOnly(y,m,d);

    }

    //overriding employee's raise method

    override public void giveRaise(double perc = 10){

      // Console.WriteLine("Child");

      Salary \*= (1+(perc/100));

      Salary += HRA+PF+DA;

    }

    //overriding Object's method

    override public string ToString(){

      return base.ToString()+"\nDate of Joining:"+joinDate+"\nDate of Retiring:"+retireDate+"\n";

    }

  }

  class test{

    public static void Main(){

      Console.WriteLine("By : Mihir L. Agrawal\n");

      Employee e1 = new Employee("James","Bond",10004.10);

      Employee e2 = new Employee("Detective","Conan",12345.50);

      Console.WriteLine("Before Raise:");

      Console.WriteLine(e1);

      Console.WriteLine(e2);

      e1.giveRaise();

      e2.giveRaise();

      Console.WriteLine("After Raise:");

      Console.WriteLine(e1);

      Console.WriteLine(e2);

      Employee pe = new PermanentEmployee("Vyomkesh","Bakshi",70000.219);

      Console.WriteLine("Before Raise:");

      Console.WriteLine(pe);

      pe.giveRaise();

      Console.WriteLine("After Raise:");

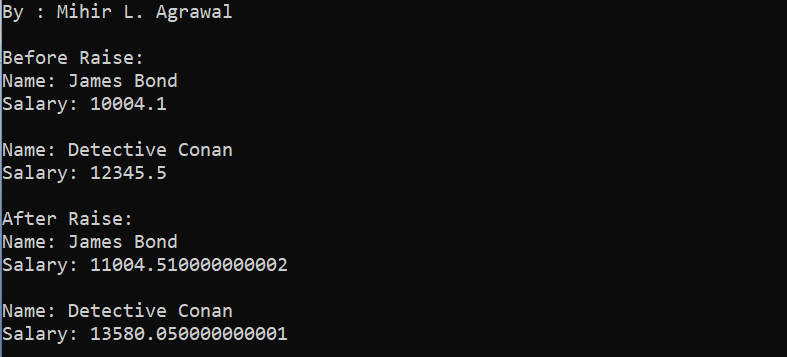
      Console.WriteLine(pe);

    }

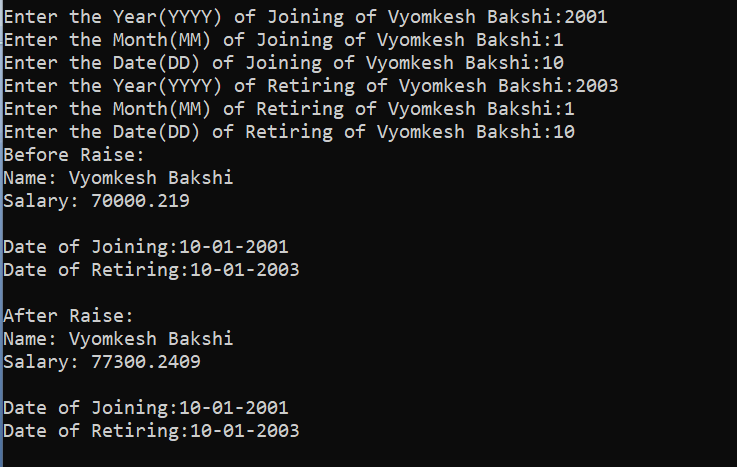
  }

}

Employee



Permanent Employee



// Read about method hiding in .net. IN your assignment also demonstrate method hiding in C#.

// REad about method overloading in C#.net . IN your assignment also demonstrate method overloading in C#.

**Overloading demonstration**

// REad about method overloading in C#.net . IN your assignment also demonstrate method overloading in C#.

using System;

namespace  EmployeeSpace{

  class Circle{

    public double radius;

    public Circle(double radius){

      this.radius = radius;

    }

  }

  class Triangle{

    public double b,h;

    public Triangle(double b,double h){

      this.b = b;this.h = h;

    }

  }

  class ShapeArea{

    //overloading area methods for different shapes

    public static double getArea(Circle c){

      return c.radius\*c.radius\*Math.PI;

    }

    public static double getArea(Triangle t){

      return t.b\*t.h\*0.5;

    }

  }

  class test{

    public static void Main(){

      Triangle t = new Triangle(3,4);

      Circle c = new Circle(10);

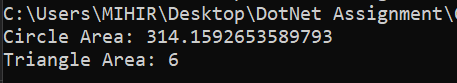
      Console.WriteLine("Circle Area: " + ShapeArea.getArea(c));

      Console.WriteLine("Triangle Area: " + ShapeArea.getArea(t));

    }

  }

}



**Method Hiding**

// Read about method hiding in .net. IN your assignment also demonstrate method hiding in C#.

using System;

namespace  EmployeeSpace{

  class Animal{

    public void walk(){

      Console.WriteLine("Animal walk on four legs");

    }

  }

  class Human: Animal{

    //below method is hiding animal's method of walk

    new public void walk(){

      Console.WriteLine("Human walk on two legs");

    }

  }

  class test{

    public static void Main(){

      Animal a = new Animal();

      a.walk();

      Human h = new Human();

      h.walk();

    }

  }

}

Output:

