Enter session in Visual Studio Code

Crear proyecto > Aplicación de consola (C#) > lab04I4 > Work with .NET 5.0

Right click Program.cs, you can see the folder where it is, so we can see: not submit just .sln. You add the whole folder as a zip

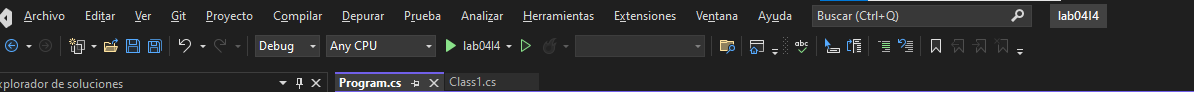
Right clock in name of the solution > Agregar > Nuevo proyecto

Biblioteca de clases> Name Point2D

Namespace is a mehcanism to isolate pieces of your code: you can rename it as TPP.Lab04, and the class as Point2D

To use the class Pint2D, click on COnsoleApp > Agregar > Referencia del proyecto > Click in Point2D

THen, you just change the namespace in ConsoleApp as TPP.Lab04 so we can use it



Switch to Release mode to send to a client

You can run with debugging or without debugging

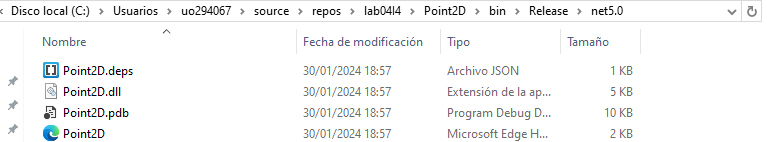
 You can save all

XML documentation

Byt typing ///

Now chabge setting: click on Point2D > Propiedades>Compilación>Salida > check Archivo de documentación

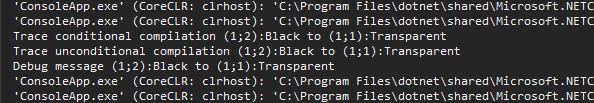
Go to Release mode and compilar solcuion just for this: compile and you will see the XML will be in the folder net5.0



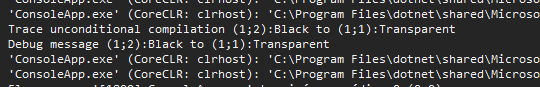
Compilar solución. Now it will complain if a method hasn’t been commented

Now run without debugging

Run with debugging: all are shown

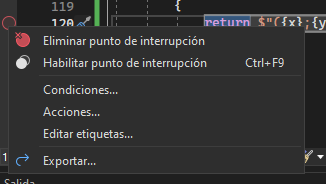


Release run debugging



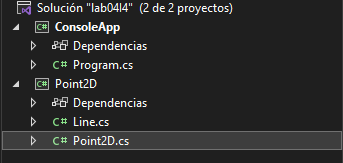


You can disable/enable breakpoints by right clicking on them.



You can add Acciones for example when y==0. It will stop execution and go to debugging mode

If the Line is external to the project, we need it to point to the Point2D



using System;

using System.Runtime.CompilerServices;

namespace TPP.Lab04

{

internal class Program

{

static void Main(string[] args)

{

Point2D a = new Point2D(1, 2, Color.Black),

b = new Point2D(1, 1, Color.Transparent),

c = new Point2D {C=Color.Red, Y=0, X=1 }; //allows us to create an object with any order of parameters

Line l = new Line(1, 2);

Console.WriteLine($"Distance from {a} to {b} is {a.Distance(b)}");

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using static System.Formats.Asn1.AsnWriter;

namespace TPP.Lab04

{

public class Line

{

/// <summary>

/// m is the slope and n the intercept

/// </summary>

double m, n;

public double M

{

get

{

return m;

}

set

{

m = value;

}

}

public double N

{

get

{

return n;

}

set

{

n = value;

}

}

/// <summary>

/// Default constructor

/// </summary>

public Line() { }

/// <summary>

///

/// </summary>

/// <param name="m"></param>

/// <param name="n"></param>

public Line(double m, double n)

{

this.m = m;

this.n = n;

}

/// <summary>

///

/// </summary>

/// <param name="p">It's the point that will be contained by the perpendicular line</param>

/// <returns>returns a Line perpendicular that contains the Point2d parameter</returns>

public Line perpendicular(Point2D p)

{

double m = -1 / this.m;

double n = p.Y + m \* p.X;

return new Line(m, n);

}

/// <summary>

/// </summary>

/// <param name="l">It's another line</param>

/// <returns>the point of the intersection as transparent</returns>

public Point2D intersection(Line l)

{

double y = (n / m - l.N / l.M) / (1 / m - 1 / l.M);

double x = (y - l.N) / l.M;

return new Point2D(x, y, Color.Transparent);

}

/// <summary>

///

/// </summary>

/// <param name="p">The point we want the distance with this line</param>

/// <returns>The distance between a point and this line</returns>

public double distance(Point2D p)

{

Line perp =perpendicular(p);

Point2D inters= intersection(perp);

return p.Distance(inters);

}

/// <summary>

///

/// </summary>

/// <returns>The represention of the line with the format: y=mx+n</returns>

public override string ToString()

{

return $"y={m}x+{n}";

}

}

}

using System;

using System.Diagnostics;

using System.Runtime.Intrinsics.Arm;

namespace TPP.Lab04

{

//In the class library define the enum Color with the values Transparent, Balck and Red

/// <summary>

/// An enumeration that represents colors

/// </summary>

public enum Color {

Transparent,Black,Red

}

/// <summary>

/// A class that represents bidimensional points

/// </summary>

public class Point2D

{

//double x { get; set; } //Automatically creates getter and setter. This syntax doesn't allow you to addlogic in getter and setter

/// <summary>

/// The coordinates

/// </summary>

double x, y; //attributes

/// <summary>

/// Color of the point

/// </summary>

Color c;

//Constructors: default constructor,

/// <summary>

/// Default constructor

/// </summary>

public Point2D() { }

//Constructor from two double (creates a Transparent point)

/// <summary>

/// Two double constructor

/// </summary>

/// <param name="x">x coordinate</param>

/// <param name="y">y coordinate</param>

public Point2D(double x, double y) {

this.x = x;

this.y = y;

c=Color.Transparent;

}

//and finally from two double and a Color.

public Point2D(double x, double y, Color color) {

this.x = x;

this.y = y;

c = color;

}

//Destructor.

~Point2D() {

Console.WriteLine($"Point2D destructor {this.x} {this.y}"); //this is no longer displayed,as the destructor is not deterministic

}

public double X //capitalize name of the attribute for comprehensive syntax of the getters and setters

{

get {

//you can add code here

return x;

}

set {

x = value;

}

}

public double Y

{

get

{

return y;

}

set

{

y = value;

}

}

public Color C

{

get

{

return c;

}

set

{

c = value;

}

}

public double Azimuth //new property that returns the angle between x and y

{

get

{

return Math.Atan2( y, x );

}

}

//• Methods: 1. Euclidean distance between two points.

/// <summary>

/// Computes the euclidean distance from the caller to the parameter

/// </summary>

/// <param name="p">the "to" point</param>

/// <returns>The distance as a double</returns>

public double Distance (Point2D p)

{

#if DEBUG //manipule source code without compiling it. You need using System.Diagnostics;

Trace.WriteLine($"Trace conditional compilation {this} to {p}"); //in Release mode this code will not be compiled. SO you cna use it, but when you release it, this line will be removed from the released code

#endif

Trace.WriteLine($"Trace unconditional compilation {this} to {p}");

Trace.WriteLine($"Debug message {this} to {p}");

return Math.Sqrt((x - p.x) \* (x - p.x) + (y - p.y) \* (y - p.y));

}

//2. ToString, a Point2d with x = 1.0, y= 2.0, c= Red is shown as (1.0;2.0):Red

public override string ToString()

{

return $"({x};{y}):{c}"; //String interpolation: what is between {} is considered an expression and will be evaluated

}

}

}