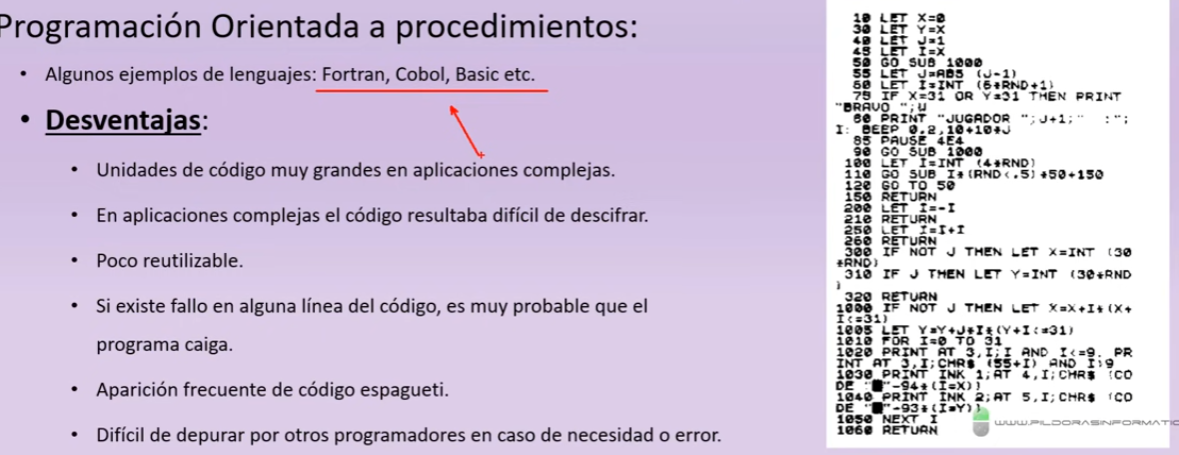
Programación orientada a objetos Videos 27-35

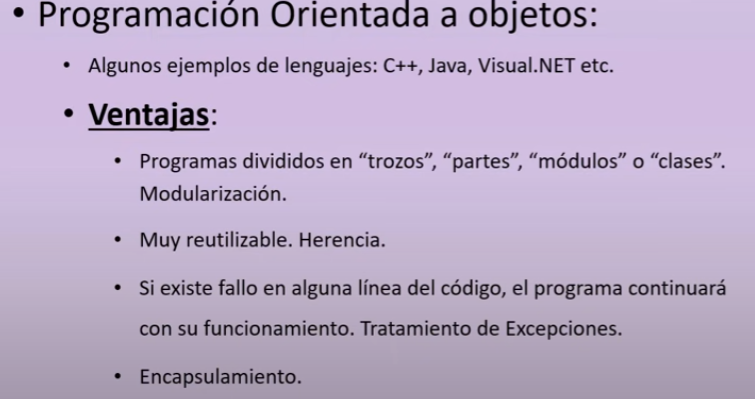
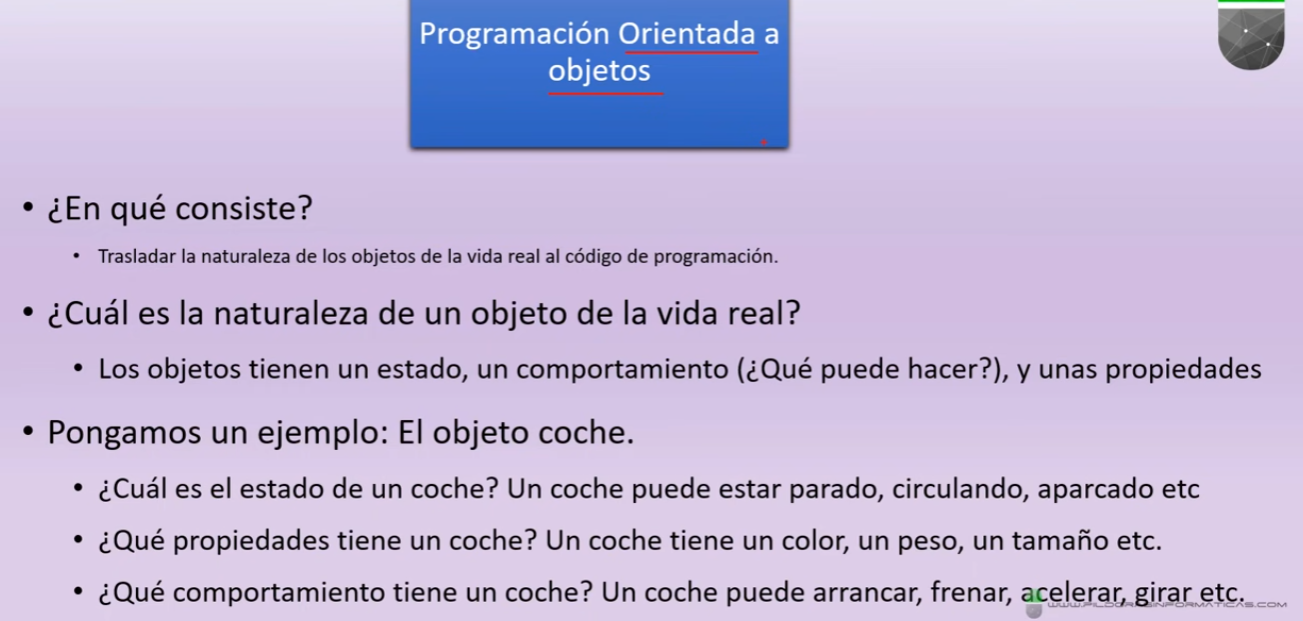
2 grandes paradigmas:

Programación orientada a procedimientos. Son los mas antiguos

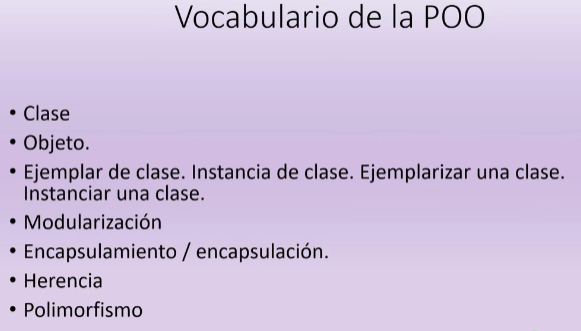
Programación orientada a objetos. Son los mas modernos.

Relativiza un poco todo esto.

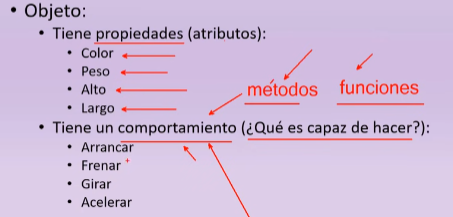




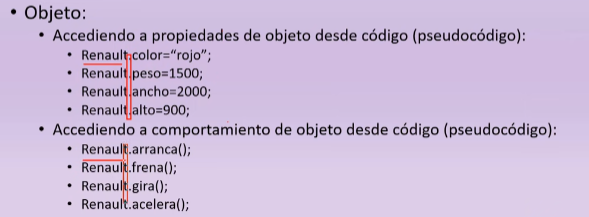
Dividir un programa en clases es modularizar. Esto hace que sea reutilizable (herencia).



Clase. Es el modelo en que se redactan las características comunes de un grupo de objetos. Es como una plantilla.



Nomenclatura del punto



Video 28

Las clases que una crea tienen que ir dentro del namespace, antes o después de la que viene dada. Dicha clase debe tener propiedades y métodos.

Clase circulo, para construir objetos círculos de diferentes áreas.

namespace POO

{

class Program

{

static void Main(string[] args)

{

Circulo miCirculo; //creación (o declaración) de objeto de tipo circulo. O una variable objeto de tipo circulo (=/= variables de tipo primitivo)

miCirculo = new Circulo(); //iniciación de variable objeto de tipo circulo. También se llama instanciar una clase. Siempre que se vea new es instanciación, o ejemplarización, o creación de ejemplar de clase.

Console.WriteLine(miCirculo.calculoArea(5)); //para que aparezca pi o calculoArea, ya que están encapsuladas, le agrega al ppio a ambas, propiedad y método, la palabra public.

//eso devuelve 78,54. Que es el area de un circulo con 5 de radio.

Circulo miCirculo2 = new Circulo(); //aquí se declara e inicializa el objeto en la misma linea.

Console.WriteLine(miCirculo2.calculoArea(9));

}

}

class Circulo

{

const double pi = 3.1416; //propiedad de la clase Circulo. Variable(o constante)=Propiedad=Campo de clase.

public double calculoArea (int radio) //Método de clase. Especifica el comportamiento del objeto.

{

return pi \* radio \* radio;

}

}

}

Video 29

Con private antes de la declaración de una variable, constante o método se encapsula, haciendo que eso solo sea accesible dentro de la clase en que se crea. Ponerlo o no es casi lo mismo, da mas claridad a la hora de leer el código.

Cuándo encapsular. Cuando en una clase se vayan a usar datos, esos tendrán que estar encapsulados, salvo que se los quiere modificar desde afuera, con un *método de acceso*. No es buena práctica ponerles el public delante.

Las variables, constantes y métodos public, deberían comenzar con mayúscula. PascalCase. public double CalculoArea (); Los que no son public va con camelCase

class Program

{

static void Main(string[] args)

{

ConversorEudoDolar obj = new ConversorEudoDolar();

// obj.Euro = 1.45; //al haberlo hecho público lo pudo cambiar. Luego lo elimina y convierte en private.

obj.CambiaValorEuro(1.45); //¿Cuánto se gana con esto?\*

Console.WriteLine(obj.Convierte(50));

}

}

class ConversorEudoDolar

{

double euro = 1.253; //este valor es volátil, cambia día a día, por eso le pone public.

public double Convierte (double cantidad)

{

return cantidad \* euro;

}

public void CambiaValorEuro (double nuevoValor)

{if (nuevoValor < 0) euro = 1.253; //\*esta línea y la que viene establece algún límite a lo que se podría modificar el valor del euro.

else euro = nuevoValor; //Esto permite cambiar el valor del euro, pero no desde afuera de la clase.

//if y else con sintaxis rara.

}

}

}

Video 30

El coche debe tener un estado inicial. Propiedades y métodos por defecto. Eso lo da el constructor.

El constructor es un método especial. Como nombre tiene el mismo que la clase obligatoriamente. No puede devolver ningún dato, pero tampoco es void (WOW!!).

namespace UsoCoche

{

class Program

{

static void Main(string[] args)

{

Coche coche1 = new Coche(); //el segundo coche es el nombre del constructor. Al que se llama.

Coche coche2 = new Coche();

}

}

class Coche

{

public Coche ()

{

ruedas = 4;

largo = 2300.5;

ancho = 800;

}

private int ruedas;

private double largo;

private double ancho;

private bool climatizador;

private String tapicería;

}

}

Luego va con los getters y setters para poder acceder a las propiedades y métodos de la clase coche.

\n Así se escribe un salto de línea.

Acá agrega el getter:

namespace UsoCoche

{

class Program

{

static void Main(string[] args)

{

Coche coche1 = new Coche(); //el segundo coche es el nombre del constructor. Al que se llama.

Coche coche2 = new Coche();

Console.WriteLine(coche2.infoCoche()); //así si, se llama al método getter, getRuedas.

Console.WriteLine(coche1.infoCoche());

}

}

class Coche

{

public Coche ()

{

ruedas = 4;

largo = 2300.5;

ancho = 800;

}

private int ruedas;

private double largo;

private double ancho;

private bool climatizador;

private String tapicería;

public string infoCoche()

{

return "Información del coche \n" + "Ruedas: "+ ruedas + ". Largo: "+ largo+ ". Ancho:"+ ancho;

}

}

Sobrecarga de constructores. Cuando se crea mas de un constructor:

Esto se hace para que los 2 autos tengan distintos estados iniciales.

Se crean varios métodos constructores, pero cada uno debe recibir distintos parámetros.

Acá agrega otro constructor y le pasa por parámetros los valores que luego se mostrarán por consola.

namespace UsoCoche

{

class Program

{

static void Main(string[] args)

{

Coche coche1 = new Coche(); //el segundo coche es el nombre del constructor. Al que se llama.

Coche coche2 = new Coche();

Coche coche3 = new Coche(4500.25, 1200.35);

Console.WriteLine(coche2.infoCoche()); //así si, se llama al método getter, getRuedas.

Console.WriteLine(coche1.infoCoche());

Console.WriteLine(coche3.infoCoche());

}

}

class Coche

{

public Coche ()

{

ruedas = 4;

largo = 2300.5;

ancho = 800;

}

public Coche (Double largoCoche, double anchoCoche)

{

ruedas = 4;

largo = largoCoche;

ancho = anchoCoche;

}

private int ruedas;

private double largo;

private double ancho;

private bool climatizador;

private String tapicería;

public string infoCoche()

{

return "Información del coche \n" + "Ruedas: "+ ruedas + ". Largo: "+ largo+ ". Ancho:"+ ancho;

}

}

Si no se le pone constructor, c# le da un constructor por defecto. Constructor que se encuentra vacío.

Video 31

Método setter (establecer). Establecen propiedades a los objetos. No tienen return, sino establecer, dar valor a las propiedades. Son void.

Normalmente un setter va acompañado de su compañero getter.

public void setExtrasCoche(bool paramClimatizador, string paramTapiceria)

{

climatizador = paramClimatizador;

tapiceria = paramTapiceria;

}

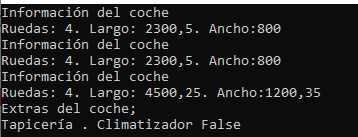
public string getExtrasCoche ()

{

return "Extras del coche: \n" + "Tapicería " + tapiceria + ". Climatizador " + climatizador;

}

Agrega eso a continuación de lo anterior. En consola devuelve esto:



Queda vacía la tapicería y es falso que haya climatizador, ya que son sus valores por defecto.

Console.WriteLine(coche2.getInfoCoche());

Console.WriteLine(coche1.getInfoCoche());

Console.WriteLine(coche3.getInfoCoche());

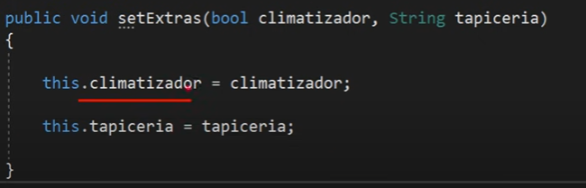
coche3.setExtrasCoche(true, "Cuero");

Console.WriteLine(coche3.getExtrasCoche());

Ahí se corrige eso, pasándole los valores por parámetro. Luego se printea y queda piola.

Recomienda luego, establecerle una tapicería por defecto. Por eso, le agrega al método constructor, tapicería=”de tela”.

Palabra reservada this. Cuando se hace referencia a un campo de clase se pone el this antes.



Para desambiguar y que el programa funcione, o bien se ponen nombres distintos a las variables, como hice yo, o le agrega antes .this.

This, refiere a esta clase. Termina recomendando el uso del this, y no como lo hice yo.

Dividir las clases en trozos. Si es compleja podría tener 2500 o 3000 lineas de código y dificultarse encontrar errores.

Se debe determinar qué debe estar en cada trozo.

Se hace con partial class mas nombre del método que se está partiendo. O sea, en la primera parte se reemplaza donde decía class Coche, se cierra y a continuación se abre de vuelta partial class Coche y se cierra al final.

namespace UsoCoche

{

class Program

{

static void Main(string[] args)

{

Coche coche1 = new Coche(); //el segundo coche es el nombre del constructor. Al que se llama.

Coche coche2 = new Coche();

Coche coche3 = new Coche(4500.25, 1200.35);

Console.WriteLine(coche2.getInfoCoche()); //así si, se llama al método getter, getRuedas.

Console.WriteLine(coche1.getInfoCoche());

Console.WriteLine(coche3.getInfoCoche());

coche3.setExtrasCoche(true, "Cuero");

Console.WriteLine(coche3.getExtrasCoche());

}

}

partial class Coche

{

public Coche()

{

ruedas = 4;

largo = 2300.5;

ancho = 800;

}

public Coche(Double largoCoche, double anchoCoche)

{

ruedas = 4;

largo = largoCoche;

ancho = anchoCoche;

}

private int ruedas;

private double largo;

private double ancho;

private bool climatizador;

private String tapiceria = "De tela";

}

partial class Coche

{

public string getInfoCoche()

{

return "Información del coche \n" + "Ruedas: " + ruedas + ". Largo: " + largo + ". Ancho:" + ancho;

}

public void setExtrasCoche(bool paramClimatizador, string paramTapiceria)

{

climatizador = paramClimatizador;

tapiceria = paramTapiceria;

}

public string getExtrasCoche()

{

return "Extras del coche: \n" + "Tapicería " + tapiceria + ". Climatizador " + climatizador;

}

}

}

Video 32

Modularización.

Cuando uno termina de trabajar un día, le gustaría cuando retome esto, recordar qué estaba haciendo. Para esto habla de comentar lo que se estaba haciendo. Puntualmente el comentario de tipo todo, que es así:

//TODO:

Menú ver. Lista de tareas. Ahí te indica donde hay comentarios de ese tipo.

Explorador de soluciones. Muestra la estructura jerárquica de lo que estamos trabajando.

Program.Cs hace referencia al fichero en el que se trabaja. Dentro está la clase program. Si se toca el triángulo que está a la izquierda, te muestra los métodos.

Se puede trabajar con varios ficheros fuente. Una nueva clase en otro fichero: se toca en el tipo de proyecto en el que trabajamos. Agregar:clase. Si no se puede ir a proyecto:agregar clase.

Así, con varios ficheros se modulariza.

Para ver si lo que uno viene haciendo está bien, se puede compilar: compilar solución.

class Punto

{

public Punto(int x, int y)

{

Console.WriteLine($"Coordenada X: {x}, Coordenada y: {y}");

}

public Punto()

{

Console.WriteLine("Este es el constructor por defecto");

}

}

Class program

static void realizarTarea()

{

Punto origen = new Punto();

}

Cuando uno establece el constructor que pide 2 parámetros, el new punto(); empieza a tirar error. Se arregla de 2 formas, o bien, poniendole los parámetros, o bien, sobrecargando el constructor, creando uno segundo que no pida parámetros, el cual sería el por defecto.

Video 33

class Program

{

static void Main(string[] args)

{

realizarTarea();

}

static void realizarTarea()

{

Punto origen = new Punto();

Punto destino = new Punto(128,80);

double distancia = origen.DistanciaHasta(destino); //el this que puse antes, dentro de distanciahasta, hace referencia a origen.

Console.WriteLine($"La distancia entre los puntos es de {distancia}");

//este método requiere un objeto para ser llamado, no como otros. Y otros son llamados por su clase. Esto haría referencia al static.

}

namespace ConceptosPOO

{

class Punto

{

public Punto (int x, int y)

{

this.x = x;

this.y = y;

}

public Punto() //este es el constructor por defecto. Cuando no se pasan parámetros.

{

this.x = 0;

this.y = 0;

}

public double DistanciaHasta (Punto otroPunto) //Este parámetro es un objeto.

{

int xDif = this.x - otroPunto.x;

int yDif = this.y - otroPunto.y;

double distanciaPuntos = Math.Sqrt(Math.Pow(xDif, 2) + Math.Pow(yDif, 2)); //teórema de pitágoras. Para eso se va a usar la clase math.

return distanciaPuntos;

}

private int x, y; //campos de clase, les llama. Propiedades, serían. Él los pone al final.

}

}

Video 34

Métodos y variables static.

Cuando se crea una variable que no es estática y se crean 3 objetos dentro de la misma clase de esa variable, se crea una copia independiente de esa variable en cada uno de los objetos creados. Luego el valor de la variable para cada uno de ellos puede variar sin afectar al resto.



Cuando se agrega el modificador static, no se crean las copias. Sino que la variable contador es *compartida* por todos los objetos que hay en esa clase. Ningún objeto la posee. *Pertenece a la clase*. Por eso se las llama variables de clase.

Objeto1.contador=9; no podría hacerse.

Ahora, si la clase es clase1, se puede clase1.contador=9;

Math.Pow () es static, pow es método de la clase math.

¿Qué se gana con esto? A veces no hay sentido para que objetos modifiquen variables.

No hace falta a las constantes ponerle que es static, C# lo asume.

namespace ConceptosPOO

{

class Program

{

static void Main(string[] args)

{

realizarTarea();

}

static void realizarTarea()

{

Punto origen = new Punto();

Punto destino = new Punto(128,80);

Punto elTercero = new Punto(23, 56);

double distancia = origen.DistanciaHasta(destino); //el this que puse antes, dentro de distanciahasta, hace referencia a origen.

Console.WriteLine($"La distancia entre los puntos es de {distancia}");

//este método requiere un objeto para ser llamado, no como otros. Y otros son llamados por su clase. Esto haría referencia al static.

Console.WriteLine($"Número de objetos creados: {Punto.retornaContador()}"); //se crearon 2 objetos, pertenecientes a la clase punto.

}

}

}

namespace ConceptosPOO

{

class Punto

{

public Punto (int x, int y)

{

this.x = x;

this.y = y;

contadorDeObjetos++; //cada vez que se cree un objeto de tipo punto, se va a incrementar en uno el valor este

}

public Punto() //este es el constructor por defecto. Cuando no se pasan parámetros.

{

this.x = 0;

this.y = 0;

contadorDeObjetos++;

}

public double DistanciaHasta (Punto otroPunto) //Este parámetro es un objeto.

{

int xDif = this.x - otroPunto.x; //estos this de acá hacen referencia al objeto desde el cual se llama al método. Es decir: double distancia = origen.DistanciaHasta(destino)

int yDif = this.y - otroPunto.y;

double distanciaPuntos = Math.Sqrt(Math.Pow(xDif, 2) + Math.Pow(yDif, 2)); //teórema de pitágoras. Para eso se va a usar la clase math.

return distanciaPuntos;

}

private static int contadorDeObjetos = 0; //Variable estática

public static int retornaContador() // método estático.

{

return contadorDeObjetos;

}

private int x, y; //campos de clase, les llama. Propiedades, serían. Él los pone al final.

}

}

Video 35

Clases anónimas. Se usan mucho para queri, y sql.

using System;

using static System.Math;

using static System.Console;

/\* double raiz = Math.Sqrt(9);

double potencia = Math.Pow(3, 4);

Console.WriteLine(raiz);

Console.WriteLine(potencia);\*/

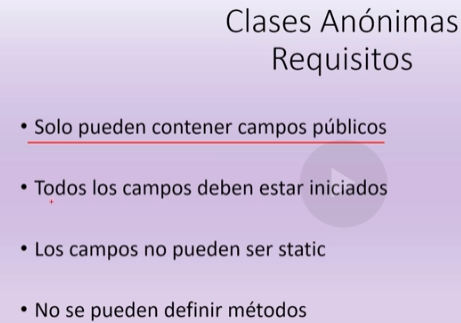
//si se usan numerosas veces los métodos de una clase, se puede importar con un using. Se puede ya no poner las Math

double raiz = Sqrt(9);

double potencia = Pow(3, 4);

Console.WriteLine(raiz);

Console.WriteLine(potencia); //Lo mismo vale para el console. Con el using podría ponerse solo el writeline. Podría llegar a confundir esto de usar los using.



Es publico por mas que no se lo declare así.

var miVariable = new { Nombre = "Juan", Edad = 19 }; //esto es una clase anónima, no se le da nombre.

Console.WriteLine(miVariable.Nombre + " " + miVariable.Edad);

//Tampoco se especifica el tipo de dato de las variables

var miOtraVariable = new { Nombre = "Ana", Edad = 25};

//El compilador afirma que pertenecen a la misma clase. Ya que tienen la misma estructura.

miVariable = miOtraVariable; //Esto lo hace para demostrar que son de la misma clase.