2. PROGRAMACIÓN BÁSICA EN C# E INTRODUCCIÓN A PROGRAMACIÓN EN AUTOCAD

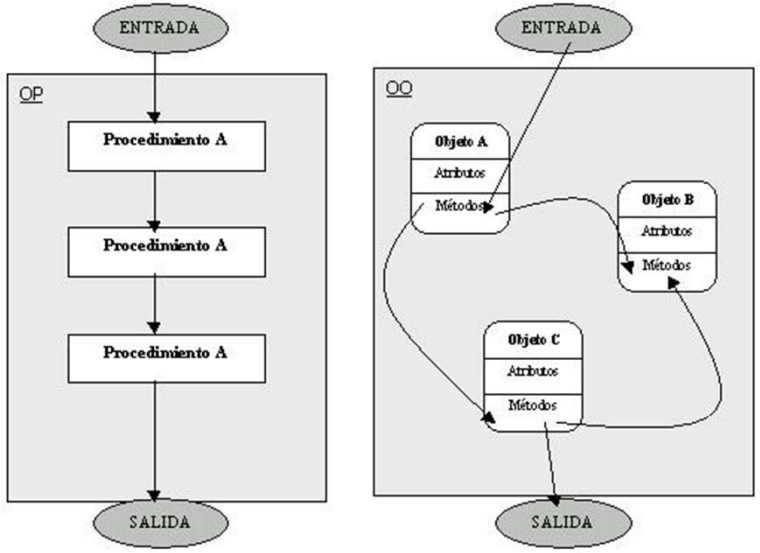
¿Cómo se estructura un programa en C#?
¿Qué son las clases y los objetos?
¿Cómo se estructuran los dibujos de AutoCAD?

Autor: Jorge A. Díez Pomares
Laboratorio de C.A.D. UMH

Índice:

2.1. CONCEPTOS FUNDAMENTALES DE C#

- C# es un lenguaje orientado a objetos.
- Está completamente estructurado en clases.
- Los algoritmos y procedimientos se ejecutan dentro de funciones, incluidas en las clases.



2.1.1. FUNCIONES: PASO DE PARÁMETROS

- Función: segmento de código con un nombre identificativo que implementa un algoritmo o rutina
- Tiene entradas y salidas
- En C# tienen que estar contenidas en una clase. Se denominan métodos.
- Declaración:
  - *permiso tipodesalida nombreFunción(tipo parámetro){}*
- Llamada:
  - *Variablesalida=nombreFunción(parámetros');*
  - No se indica el tipo de parámetro, y no tienen por que llamarse igual que en la declaración

2.1.1. FUNCIONES: PASO DE PARÁMETROS

- Entradas:
  - Parámetros de entrada
  - Variables globales
  - Atributos de la clase accesibles
- Salidas:
  - Parámetros de salida
  - Variables globales
  - Atributos de la clase accesibles
  - Variable de salida de la función: Desde tipos básicos hasta objetos complejos y listas de ellos.

2.1.1. FUNCIONES: PASO DE PARÁMETROS

- Paso de parámetros a una función:
  - Los parámetros de la función se especifican entre paréntesis a continuación del nombre de la función, con la siguiente estructura.
    - *Public tipo nombreFuncion(tipo1 a, tipo2 b,… ,tipoz z);*
  - **Paso de parámetros por valor**: Se pasa a la función una copia de la variable, las ediciones sobre ese parámetro no afectan al valor de la variable original.
    - *Public tipo nombreFuncion (tipo a){}*
  - **Paso de parámetros por referencia**: A la función se le pasa la referencia a una variable (similar al paso por dirección de C/C++), por lo que los cambios sobre el parámetro están referidos a la variable original que cambiará.

2.1.1. FUNCIONES: PASO DE PARÁMETROS

- Paso de parámetros por referencia:
  - Se declaran añadiendo la palabra *out* o *ref* antes del tipo de la variable.
    - *public tipo nombreFuncion(ref tipo a);*
    - *public tipo nombreFuncion(out tipo a);*
  - En la llamada a la función es necesario especificar también la palabra *ref* u *out* según el caso.
    - *Variablesalida= nombreFuncion(ref b);*
    - *Variablesalida= nombreFuncion(out b);*
  - Diferencia entre ref y out:
    - ref: La variable debe estar inicializada al llamar a la función.
    - out: No requiere que la variable esté inicializada al llamar a la función. Pero sí requiere que la función la inicialice.

2.1.1. FUNCIONES: PASO DE PARÁMETROS

- Ejemplo:
  - Declaración:

```
public int funcionEjemplo(int valor, ref int refer, out int salida){
    refer=valor;
    valor=5;
    salida=valor;
    int a=valor;
    return a;
    }
```

  - Llamada:

```
    int a,b,c,d;
    ……¿Qué dos líneas tendría que poner aquí?
    a=funcionEjemplo(b, ref c,out d);
```

¿Qué valor toman a,b,c,d al terminar de ejecutarse la función?

2.1.2. CLASES Y OBJETOS: MÉTODOS, ATRIBUTOS, HERENCIA Y PERMISOS

- ¿Qué es una clase?
  - Es el modelo de un concepto. Delimita los estados y comportamientos posibles de los casos concretos de la clase.
  - El estado viene encapsulado por los Atributos.
  - El comportamiento viene encapsulado por los Métodos.
- ¿Qué es un objeto?
  - Es un caso particular del concepto al que se refiere la clase. Es decir, tiene unos atributos particulares y un comportamiento asociado concreto. Un objeto creado a partir de cierta clase se denomina instancia de esa clase.

2.1.2. CLASES Y OBJETOS: MÉTODOS, ATRIBUTOS, HERENCIA Y PERMISOS

- Atributos:
  - Son variables propias de la clase que sirven para definir sus características globales.
  - Para cada objeto los atributos pueden tener distintos valores, pero todos ellos tendrán los mismos atributos.
  - Se pueden asignar valores por defecto a tales atributos.
- Métodos:
  - Son funciones que definen el comportamiento de la clase.
  - A partir de entradas externas y el estado actual del objeto (atributos) generan una respuesta en forma de variable de salida y cambio del estado del objeto.
  - Unos métodos específicos e indispensables de cualquier clase son los CONSTRUCTORES, que son los encargados de crear las instancias de la clase. Se distinguen del resto de métodos porque su nombre coincide con el de la clase.

2.1.2. CLASES Y OBJETOS: MÉTODOS, ATRIBUTOS, HERENCIA Y PERMISOS

- Sobrecarga de funciones:
  - Varios métodos de un mismo objeto pueden tener el mismo nombre.
  - Se diferencian entre ellos por el número y tipo de argumentos.
  - El compilador sabrá a que método llamar según los parámetros pasados a la función.
  - Práctica habitual.
- Sobrecarga de operadores:
  - No es tan habitual pero puede resultar útil.
  - Investigad sobre ello.

2.1.2. CLASES Y OBJETOS: MÉTODOS, ATRIBUTOS, HERENCIA Y PERMISOS

- Herencia:
  - Una clase puede heredar de otra clase.
  - Heredar quiere decir que la clase heredera (subclase) tiene implícitamente definidos métodos y atributos de la superclase (No todos).
- Especificadores de acceso:
  - Se aplican a atributos, métodos y clases. Afectan a su visibilidad por otras funciones y a la herencia.
  - Public: Método o atributo accesible desde todas las funciones. Los miembros public se heredan como public.
  - Protected: Método o atributo accesible sólo por los métodos de la propia clase. Los miembros protected se heredan como private.
  - Private: Método o atributo accesible sólo por los métodos de la propia clase. Los miembros private no se heredan.

2.1.2. CLASES Y OBJETOS: MÉTODOS, ATRIBUTOS, HERENCIA Y PERMISOS

- Ejemplo de clase:

```
public class ClaseA:ClaseB{
        //Atributos
        public int a;
        private char b;
        protected double c;

        //Constructores
        public  ClaseA(){
        }
        public ClaseA(int a){
                this.a=a;
        }
        //Métodos
        public void metodo1(double b){
                metodo2(b);
        }

        private void metodo2(double d){
                c=d;
        }
}
```

2.1.2. CLASES Y OBJETOS: MÉTODOS, ATRIBUTOS, HERENCIA Y PERMISOS

- Creación y uso de objetos:
  - Los objetos se declaran como una variable de tipo la clase.
    - *claseA objeto1;*
  - Se inicializan utilizando uno de los constructores y el operador new.
    - *objeto1=new claseA();*
  - Acceso a atributos y métodos: operador .
    - *objeto1.a* :Accedemos al atributo “a”
    - *objeto1.metodo1(parametro)* :Accedemos al metodo1

2.1.2. CLASES Y OBJETOS: MÉTODOS, ATRIBUTOS, HERENCIA Y PERMISOS

- Clases, atributos y métodos static:
  - Podemos declarar una clase, atributo y método como static.
  - Esto implica que esa clase, atributo o método puede ser utilizada sin tener que crear una instancia de ella.
  - En las clases static no se pueden crear instancias e ellas.
  - Si se modifica un atributo static se modifica para todos los objetos de la clase ya creados y por crear.
  - Por tanto para acceder a un atributo o método tipo static usamos:
    - *claseA.atributoestatico*
    - *claseA.metodoestatico*
  - Ejemplos de clases muy utilizadas con método static:
    - *System.IO.Path* (Clase estática con métodos estáticos para trabajar con rutas)
    - *System.IO.File* (Clase estática con método estáticos para trabajar con ficheros)

2.2. VISUAL STUDIO: HERRAMIENTAS DE DEPURACIÓN

- Depurar: Ejecutar el código de forma controlada, añadiendo puntos de interrupción y comprobando el valor de las variables, con el objetivo de localizar errores en el código.
- VisualStudio: Herramientas potentes de depuración.
  - Punto de interrupción: El código se ejecuta hasta la línea especificada, donde se detiene para que el programador efectúe las comprobaciones necesarias.
  - Ejecución línea a línea: Es posible ejecutar el programa línea a línea, pudiendo decidir si entrar en el código de las subrutinas (F11) o no (F10).

CASO PRÁCTICO

- Definir la clase HéliceCuádrica, que iremos ampliando a lo largo del curso.
  - Atributos:
    - Radio
    - Paso
    - Incremento angular (definirá la precisión)
    - Coordenadas de cada punto de muestreo
  - Constructores:
    - Constructor dado el radio, paso e incremento
    - Constructor copia
    - Constructor por defecto
  - Métodos:
    - Escribir Script que dibuje la spline que aproxima la hélice
    - Cálculo de la longitud de la hélice
    - Recalcular modificando radio o paso o ambas (sobrecarga)
    - Cambiar precisión, recalcula pero con incremento angular

CASO PRÁCTICO

- Ayudas:
  - Fórmulas:
    - Radio en polares=Radio*cos(ángulo vertical en radianes)
    - Z=Paso/360*ángulo horizontal (en grados)
    - Ángulo vertical=ASIN(Z/Radio)
  - Librería Matemática básica: Math (incluida en System), de tipo estático.