1

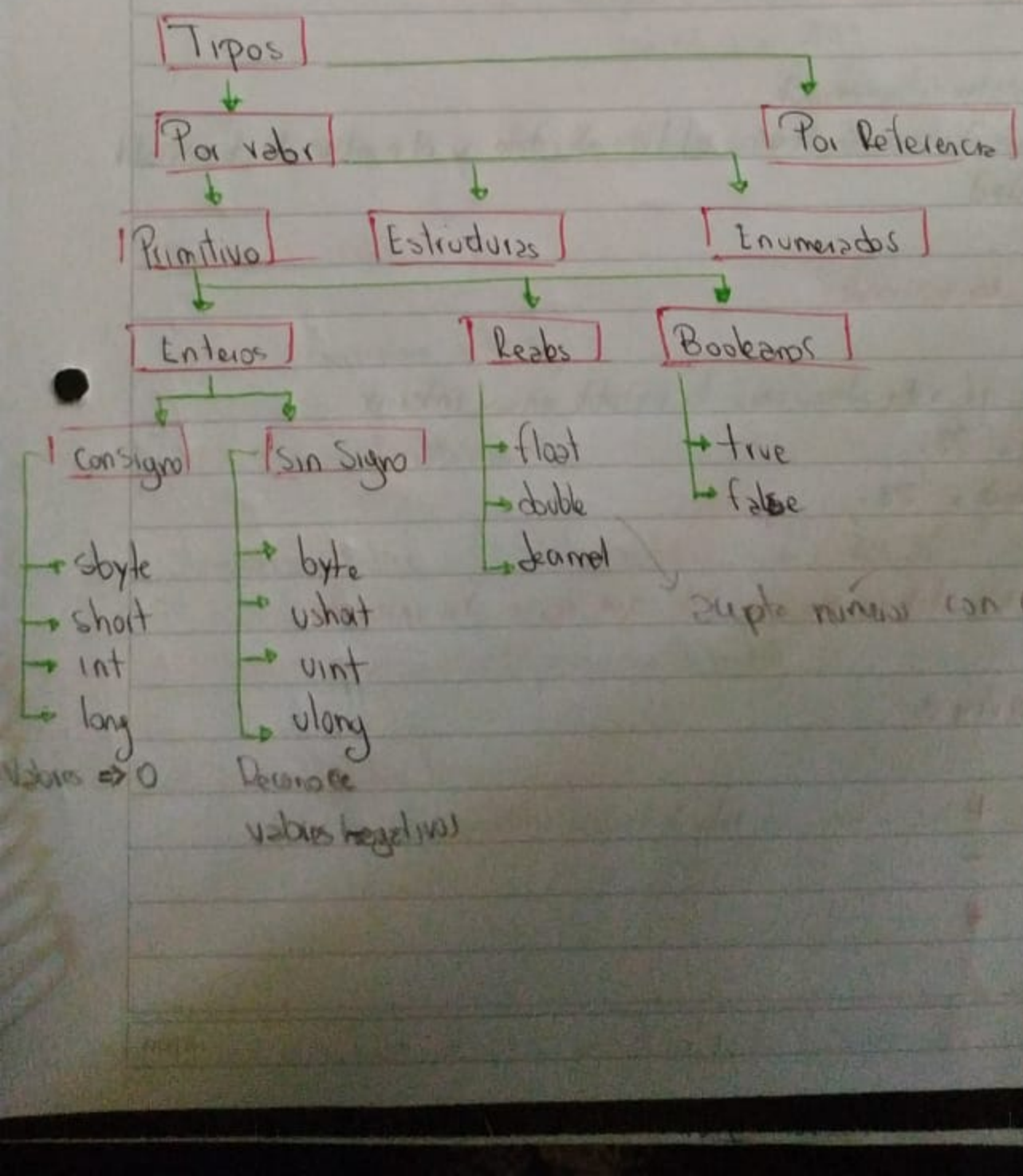
CURSO C#

Los métodos no terminan en ;
Las instrucciones terminan en ;

Console . WriteLine
Instancia Método

Clases Propias

Clases Predefinidas
↓
Biblioteca de clases API



Variable

Espacio en la memoria (RAM) del ordenador donde se almacenará un valor que podrá cambiar durante la ejecución del programa

Buenas Prácticas

- No comenzar el nombre de una variable con guion bajo → -edad
- No crear más de una variable que se diferencien sólo por una letra
- Comenzar el nombre de la variable con letra minúscula
- Si el nombre de la variable está compuesto por más de una letra comenzar la segunda con mayuscula (edadAdulto) (camelCase)
- No utilizar notación húngara → ~~iedad~~
  int → tipo de dato

¿Qué es definir una variable?

- es crear / construir la variable
- en C# para declarar se indica el tipo de dato y el nombre de la variable
  int edad

¿Qué es iniciar una variable?

- es especificar qué valor almacenará la variable en su interior
  - edad = 28.
  - int edad = 28;

En C# no se puede utilizar una variable que no se ha iniciado

Operadores Aritméticos

Suma	+ se utilizan con valores de texto (cadenas strings)
Resta	-
Multiplicación	*
División	/
Residuo / Módulo	% devuelve el resto de una división, no de un porcentaje

edad = 19

edad += 5 → edad = 24

++ incremento
-- decremento

+=5 → incremento en valor 5
-=9 → decremento en valor 9

El programa lee de Arriba hacia Abajo ↓ y de izquierda a Derecha →

edad = 19 ↓

cw(" edad++ ") = 19

(" ++edad") = 20

Operador

igual/asignación = edad1 = edad2 = 29

↓ asignación ↓ igual

Declaración Implícita de variable

var asigna la variable automáticamente

var edad = 27;

↑ int → le asigna el tipo

nota: el valor de la variable no podrá cambiarse

Conversión explícita

double temperatura = 34,5;

int temperaturaMadrid;

temperaturaMadrid = (int) temperatura → 34,5

conversión implícita (conversiones directas)

int → double → long

Conversión texto → número

Int.Parse( )
↓
Tipo.Parse

Constantes

Espacio en la memoria del ordenador donde se almacenará un valor que no podrá cambiar durante la ejecución del programa

const int valor = 5; • debes inicializarse en la misma línea

• es recomendable escribir el nombre de la constante en mayúscula

Metodo Math
.Pow (base, exponente). Potencia

Metodos

¿Qué son?
Grupo de sentencias (instrucciones) a las que se les da un nombre identificativo, que realiza una tarea en concreto

¿Para qué sirven?
Para realizar una tarea en concreto en un momento determinado. Un método no realiza su tarea hasta que no es llamado

¿Cuál es su sintaxis?

. TipoDevuelto nombreMetodo(Parámetros)
{ Cuerpo del método
}

(3)

- No se ejecutará hasta ser llamado
- Todos los métodos van dentro de una clase
- Se debe identificar el tipo devuelto y los parámetros
- No hay distinción entre métodos y funciones. En C# son lo mismo
- Un método void jamás llevará la instrucción return
- Un Método puede ser llamado la veces que queramos
- Cuando se trabaja con tipo de dato primitivos hay que tener en cuenta que tipo de datos va a devolver y que parámetros estamos pasando

- static (int) Sum ((double num1, int num2))

* Salta error porque no se puede transformar implícitamente double en int

Si un método contiene "una sola línea de código" se puede escribir o representar de la sig. manera:

static double Sum (double N1, int N2) => N1 + N2

expression-bodied

{

Todo el contenido de las llaves
se denomina Contexto, ámbito o alcance

El contenido tiene una visibilidad local.
Solo existe dentro del método

{

Son accesibles por el método si inicializamos las variables dentro de ámbito de la clase

Por ej: class Program
{
int n1=5;

- Las variables definidas en el ámbito de clase reciben el nombre de "campo de clase"
- Pueden colocarse al final del script o al inicio

Sobrecarga de métodos

Es cuando hay 2 métodos en el mismo ámbito con el mismo nombre.

Para que se puede hacer los métodos deben recibir distintos tipos de parámetros o diferentes cantidades de parámetros

Parámetros opcionales

Se usan en el caso de que un programa no soporte sobrecarga

```
static double suma (double n1, int n2, int n3=0)
  {
  }
```

obligatorios — opcional

Cuando se llama al método puedo pasar 2 variables y le opcional valdrá cero. O puedo pasar 3 y entonces la variable opcional tendrá un valor

Los parámetros obligatorios siempre van al inicio y después van al final los opcionales, nunca puede ser al revés.

Se da el caso en que haya una ambigüedad, en este caso 2 métodos "iguales". Por ej:

```
Console.WriteLine (suma(5,7,2));
double Suma (int n1, double n2, double n3=0) { }
double Suma (int n1, double n2) { }
```

Llamará al 2do método porque es el que más se adapta al número de parámetros.
A pesar de que la llamada al método pueda hacer referencia al 1er método

Condicional IF

- Valores booleanos (bool) → true
  → false

- ! → operador negación

Por ej: bool hacefrio = true;
!haceFrio → false

Operadores de comparación

== → igual que
!= → diferente que
< → menor que
<= → menor o igual que
> → mayor que
>= → mayor o igual que

Operadores Lógicos

&& → Y lógico
|| → O lógico

Al If, DO y a los demás bucles se les denomina como "Estructuras de control de flujo"

```
If(condicion)  → true
{              → false
.
.
}
```

Si el If contiene solo una línea se pueden eliminar las llaves

Nota: Solo se tomará la primera línea después del If

```
If(condicion)
    └→ xxxxx  → Solo tomará esta línea
       xxxx
```

Estructura Else-if

Se utiliza a la hora de evaluar varias condiciones encadenadas

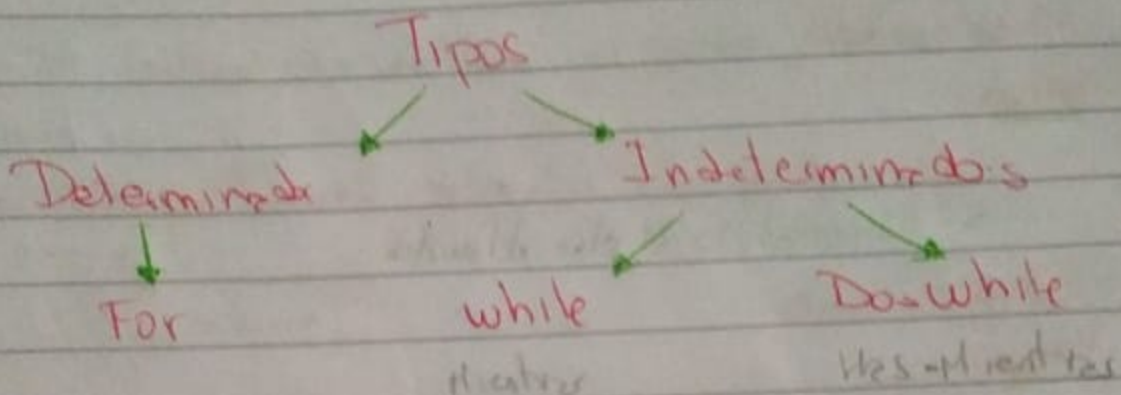
Ver vídeo 18

Condicional switch

Todo lo que podemos hacer con un switch lo podemos hacer con un if. "Casi" todo lo que podemos hacer con un if, lo podemos hacer con un switch

Ventajas

- Permite repetir código de forma rápida y sencilla
- Ahorro de tiempo y horas de programar
- Permite trabajar con grandes volúmenes de datos
- Permiten trabajar con arrays



Do-while

- Se ejecutará al menos una vez

Excepciones

Son errores en tiempo de ejecución del programa que escapan al control del programador

Try intenta
Catch captura

```
try
{ código que intenta ejecutar
}catch( FormatException ex )
{ código que se ejecuta si hay excepción
}
```

Tipo de excepción — nombre

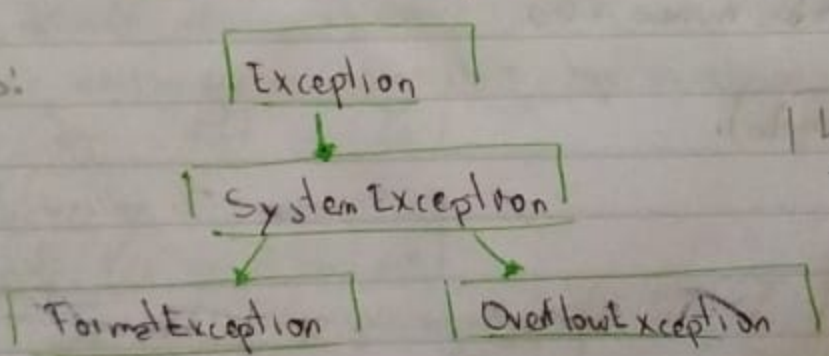
Varias excepciones

```
try
{
}
catch (FormatException ex)
{
}
catch (OverflowException ex)
{
}
```

→ para números muy grande / o cuando introduce texto (creo)

Herencia de Excepciones

Ejemplo:



En general, hay que ser preciso al determinar una excepción

Nota:

- Después de usar un captura general con "Exception" no puedes utilizar una clase que herede de "Exception"

```
catch(Exception)
{
}
catch(FormatException) → Error
```

- Primero, van los específicos y, el último, los generales

Filtro de Excepciones

```
catch (Exception e) when (e.GetType() != typeof(FormatException))
```

Que tome todas las excepciones que no sean del Tipo FormatException

Uso de checked

```
checked
{
    int numero = int.MaxValue
    int resultado = numero + 20

    Console.WriteLine(resultado);
}
```

tendría que dar un error porque excede el valor número que puede contener el tipo int. pero el programa da un resultado erróneo pero que sigue ejecutando

Con el checked le damos al programa que revise el código para que lance la excepción

* Para que el programa lo haga automáticamente hay que ir a:
  - Click izquierdo sobre la solución, Propiedades
  - compilación
  - Avanzado (el último)
  - Comprobar desbordamiento aritmético

Como configurar excepciones

- Depurar
  - Ventanas
    - Configurar excepciones

Bloque finally

```
try
{
    // código que se intenta ejecutar
}
catch (FormatException e)
{
    // código que se ejecuta si hay excepción
}
finally
{
    // código que se ejecuta siempre
}
```

- Se utiliza en Base de datos, lectura de ficheros externos, etc

Programación orientada a objetos POO

Paradigmas de la Programación

- Programación orientada a Procedimientos (juego)
- Programación orientada a objetos (modelo)

¿En qué consiste?

Trasladar los objetos de la vida real al código de programación

¿Cuál es la naturaleza de un objeto de la vida real?

Los objetos tienen un estado, un comportamiento (¿Qué puede hacer?) y unas propiedades

Pongamos un ejemplo: El objeto coche

- ¿Cuál es el estado de un coche? Un coche puede estar parado, circulando, aparcado, etc
- ¿Qué propiedades tiene un coche? Un coche tiene un color, un peso, un tamaño, etc
- ¿Qué comportamiento tiene un coche? Un coche puede arrancar, frenar, acelerar, girar, etc

Ventajas

- Programas divididos en trozos
- Muy utilizable. Herencia
- Si existe un fallo en alguna línea de código, el programa continuará con su funcionamiento. Tratamiento de excepciones
- Encapsulamiento

PUBLIC

accesible desde cualquier parte

PRIVATE

accesible desde la propia clase

PROTECTED

accesible desde la clase derivada

INTERNAL

accesible desde el mismo ensamblado

8

PROTECTED INTERNAL

accesible desde el mismo ensamblado o clase derivada de otro ensamblado

PRIVATE PROTECTED

accesible desde la misma clase o clase derivada del mismo ensamblado

POR DEFECTO

accesible desde el mismo paquete

Clase

Es como una plantilla.

Por ej: las características comunes que tienen los coches podríamos llamarlas clase: el chasis, capó, motor, vault, ruedas, luces

Objetos

Son los que derivan de esa clase o plantilla

Por ej: con todas esas características: capó, ruedas, motor, luces, etc. puedo crear un determinado coche (objeto) y cada coche (objeto) tienen sus propias características color diferente, carrocería diferente

Auto.
- Tiene propiedades (atributos)
  - color
  - peso
  - largo
  - alto
- Tiene un comportamiento (¿Qué puede hacer?) ← Métodos o funciones
  - frenar
  - frenar
  - girar
  - acelerar

Cuando declaramos una variable en una clase estamos definiendo una propiedad

Encapsulación

Es una medida de seguridad y buen funcionamiento de nuestro programa
Hace que los datos de una clase no sean accesibles desde otra

colocando la palabra private estamos encapsulando

Si en una clase vas a usar constantes o datos que no se deben modificar se tiene que encapsular

Convenciones

- Los identificadores "public" deben comenzar con letra mayúscula
  - Notación "PascalCase"
    - Ej: public double CalculoArea(){...
- Los identificadores que no son "public" deben empezar por letra minúscula
  - Notación "camelCase"
    - Ej: longitudRadio

9

Constructores Video 30

Finalidad, dar un estado inicial a todos los objetos por ej: de tipo cache.

```
public cache()
{ ancho=4;
  ancho= 300,5;
}
```

Getters y Setters

this sirve para diferenciar cuando me refiero a un campo de clase y cuando a un Parámetro

Variables y Métodos static

Ejemplo:

```
static int contador = 7;
```

nadie puede modificarlo, pero pueden usarlo. Solo la clase que contiene la variable estática puede modificarlo

Video 34 quedé

```
switch (expresión de control) {
    case expresión constante:
        codigo a realizar
        break;
    case expresión constante:
        codigo a realizar
        break;
    default:        (Adicional) es el else del switch
        codigo a realizar
        break;
}
```

- Cada expresión cte ha de ser única
- Solo se puede usar el switch para evaluar:
  - int
  - char
  - string
  - (float y double han de utilizar if)
- Las case solo pueden contener expresiones ctes
- Todas las cases deben llevar su break
  - Se puede utilizar return y throw.

BUCLES

Permiten repetir la ejecución de líneas de código un número determinado o indeterminado de veces.