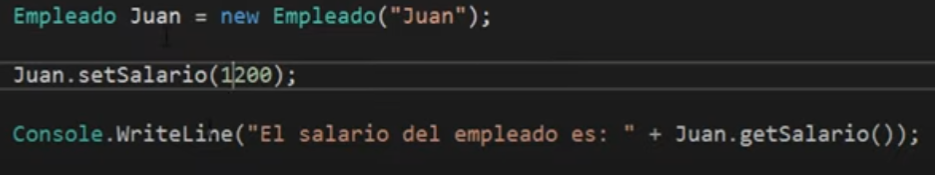
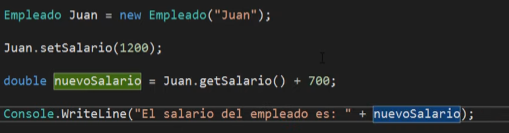
Propiedades

Sería una alternativa a la encapsulación.



Repaso tranqui de lo que es set y get



Ahí se le agregó un plus



De esa forma no puede hacerse, ya que el salario está encapsulado, es private.

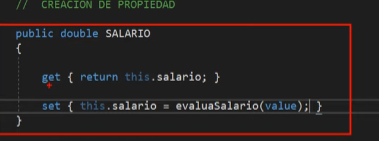
Tampoco es correcto hacer public el salario de juan.

Las properties reúne lo mejor de los 2 mundos: de la encapsulación, y de una sintaxis sencilla. (algo así).

Permiten acceder como si fuese public pero no permite violar reglas que no queremos que se violen.

Se pone todo el nombre en mayúsculas. Va un get y un set, dentro de la propiedad.

EvaluarSalario es el método al que se le pasa por parámetro el salario y evalua si es o no mayor que 0.



Luego, en main se pone así, sencillo:

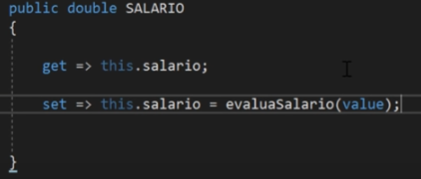




Así se le da el plus.

Si se le quisiera poner de salario -750, printearía que el salario es 0, ya que el método evaluasalario lo establece en 0.

Expresion boddied con el operador lambda (=>). Simplifica un poco mas la escritura:



Para despejar dudas sobre sintaxis, se le pone a los campos de clase un guion bajo y al properties no, queda todo en mayuscula.

Hay properties de solo lectura y de solo escritura.

Con solo escritura se prescinde del get, y de solo lectura se prescinde del set.

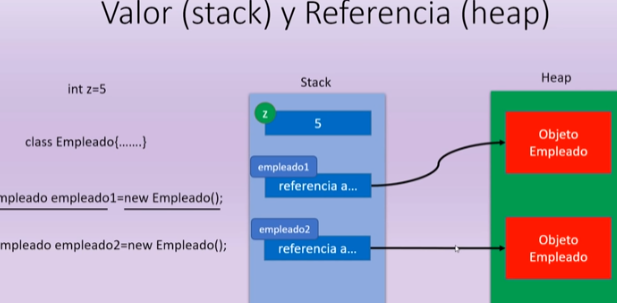
Estructuras-Structs

Las clases se almacenan en la memoria heap y las structs en la memoria stack.

Hay 2 tipos de memoria cuando estamos creando una aplicación.

Los tipos primitivos van a Stack. Int z=5;

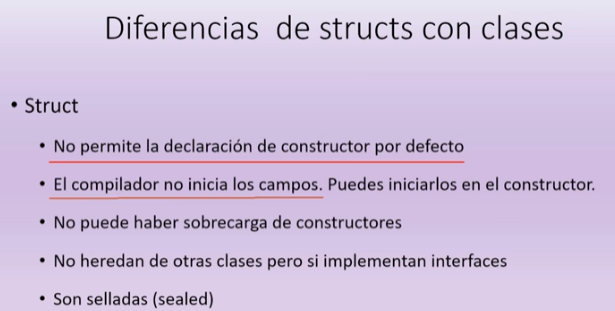
Cuando se crea un objeto eso va a heap. La referencia se crea en el stack.

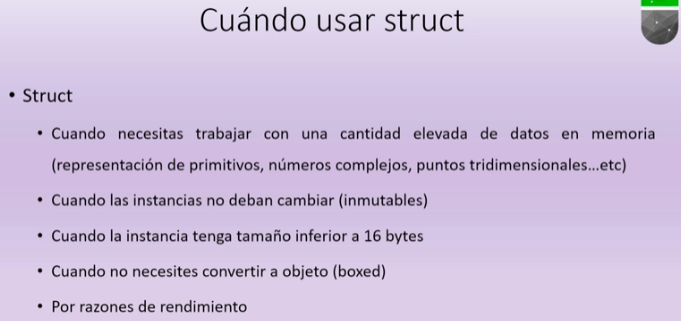


Cuando se instancia un struct no va al heap tampoco, siempre se maneja en el stack, creándose copias, no hay referencias.

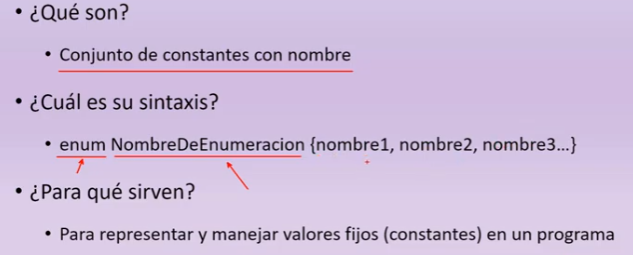
A primer vista funcionan parecido, se reemplaza simplemente class por struct.

La stack es de acceso mas rápido, a la vez que es temporal. Cuando el método termina su ejecución, la información se pierde. Esto no es así en el heap.





Enum (Tipos enumerados)



Supongase que quiero representar las estaciones del año, o días de la semana, o meses del año. Son series con valores constantes. En estos casos es mas cómodo y conveniente.

Lo mas común es verlo dentro del namespace, ya que lo habitual es que se use en mas de una clase.

Se puede almacenar un valor nulo dentro de un tipo enumerado.

Va el nombre de tipo enumerado ? variable donde se guardara=null;



namespace Enum

{

class Program

{

static void Main(string[] args)

{

Estaciones Alergia = Estaciones.Primavera; //esto es lo segundo que hace.

Console.WriteLine(Alergia);

//String LaAlergia=Estaciones.Primavera //Esto no es correcto, ya que estacioens no es un string

string LaAlergia = Alergia.ToString(); //esto si sería correcto

Console.WriteLine(LaAlergia);

}

}

enum Estaciones { Primavera, Verano, Otoño, Invierno}; //Se les asigna un valor numérico, comienza por 0. Es modificable.

}



Se les puede asignar otros valores.

static void Main(string[] args)

{

Bonus Antonio = Bonus.bueno;

Console.WriteLine(Antonio); //Esto da, pero ¿como se imprime el valor?

double BonusAntonio = (double)Antonio; //Habla de casting.

Console.WriteLine(BonusAntonio); //ahora si.

double SalarioAntonio = 1500 + BonusAntonio;

Console.WriteLine(SalarioAntonio);

}

}

enum Bonus { bajo=500, normal=1000, bueno=1500, extra=3000}

class Program

{

static void Main(string[] args)

{

Empleado Jauria = new Empleado(Bonus.extra, 1900.60);

Console.WriteLine("El salario de Jauria es: "+ Jauria.getSalario());

}

}

class Empleado

{

public Empleado (Bonus BonusEmpleado, double Salario)

{

Bonus = (double)BonusEmpleado;

this.Salario = Salario;

}

public double getSalario()

{

return Salario + Bonus;

}

private double Salario, Bonus;

}

enum Bonus { bajo=500, normal=1000, bueno=1500, extra=3000}

}

Destructores

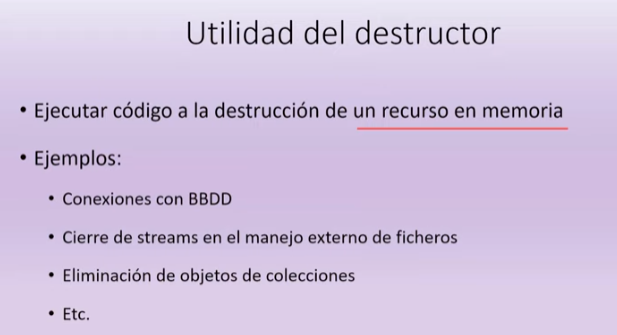
Stack y Heap

Empleado pedrito=new empleado;

Esto crea una referencia en Stack y se crea el objeto en Heap.

Hay objetos que ya dejan de utilizarse, ya que no hay mas referencias al mismo. Ahí el recolector de basura (garbage collection) podría eliminar el objeto, así se ahorran recursos de la pc.

En c# el garbage collection es automático, en otros lenguajes, mas antiguos, no.



Método destructor, ejecuta código cuando el garbage collection elimina un recurso. El garbage elimina al recurso y los recursos asociados, aunque no siempre es inmediato.

BBDD (base de datos).

class Program

{

static void Main(string[] args)

{

ManejoArchivos miArchivo = new ManejoArchivos();

miArchivo.Mensaje();

}

}

class ManejoArchivos

{

StreamReader archivo = null;

int contador = 0;

string linea;

public ManejoArchivos()

{

archivo = new StreamReader(@"C:\Users\Pcablo\Desktop\Programando\C#\tirar.txt");

//arriba se abre la conexión, flujo de datos. Si no se cierra, una vez que se leyó, consume recursos.

while ((linea = archivo.ReadLine()) != null)

{

Console.WriteLine(linea);

contador++;

}

}

public void Mensaje()

{

Console.WriteLine("Hay {0} lineas", contador);

}

//Alt 126 ~

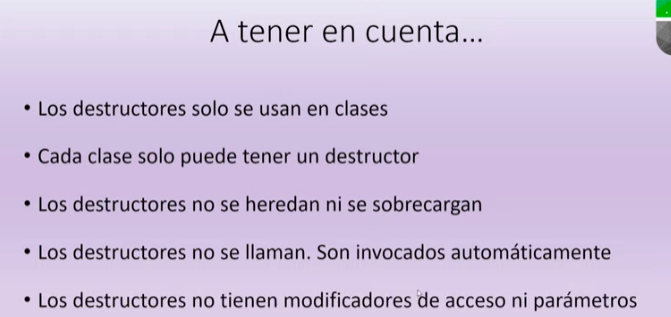
~ManejoArchivos() //es igual el nombre que el constructor

{

archivo.Close(); //Ahí se cierra el flujo de datos.

}

}



Recomienda dejar usar el garbage collection y solo usarlo al destructor cuando tenes muy claro que tenes que eliminar algo jodido. Es fácil equivocarse y dificulta la lectura del código.