

# Hada T9: Bibliotecas

---

Herramientas Avanzadas para el Desarrollo de  
Aplicaciones

# Objetivos del tema

- Saber qué es una biblioteca. Aprender a crearlas y usarlas.
- Saber qué son las referencias en un proyecto C#.
- Conocer y saber usar los proyectos de tipo *Librería* en C#+VisualStudio.

# ¿Qué es una biblioteca?

- De manera muy resumida podemos decir que una *biblioteca* -A lo largo del tema emplearemos el término *biblioteca* en lugar de *librería* por ser el primero más apropiado.- **ES UN** compendio de recursos (normalmente binarios): *subprogramas, clases, datos, iconos, etc...*
- **Cuando distribuimos estos recursos dentro de una biblioteca estamos favoreciendo su uso y reutilización.**
- ¿Motivo?: En el caso de *código fuente* no es necesario recompilar ya que éste se distribuye dentro de la biblioteca en forma binaria, ya compilado; hasta ahora solo sabíamos redistribuirlo en forma de código fuente.
- **Para emplear una biblioteca hemos de *enlazar* nuestro código con dicha biblioteca,** de esta forma tenemos acceso a su contenido.

# ¿Por qué distribuir algo en formato binario?

- Si para usarlo debe estar en formato binario, le evitamos al usuario del mismo tener que generar este formato binario a partir de ‘sus fuentes’.
- En ocasiones el proceso de compilación y obtención de una biblioteca es costoso y puede que no sea sencillo.
- En el caso de las **bibliotecas de enlace dinámico** tenemos la ventaja de poder cambiarlas para solucionar problemas sin necesidad de recompilar.
- Aunque estas bibliotecas en ocasiones son fuente de numerosos problemas, echa un vistazo al concepto de [DLL Hell](#).

# Bibliotecas estáticas vs dinámicas

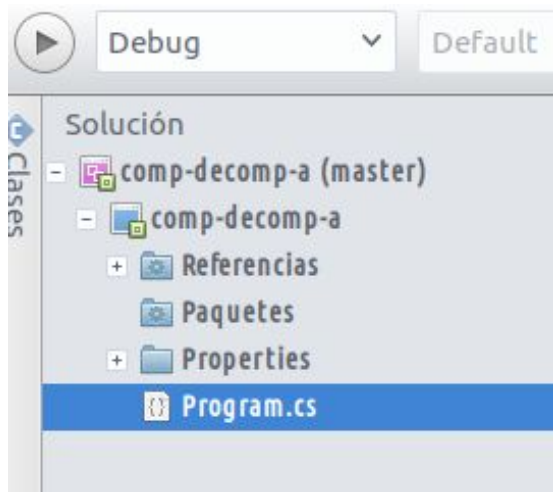
- **Estáticas:** *(unix extensión .a, win extensión .lib)*
  - El código de la librería se adjunta al ejecutable, que incrementa de tamaño. *(compile time)*
  - *Ventajas:*
    - *No hay problemas de dependencia.*
    - *Mejoras en el rendimiento*
    - *Simple distribución e instalación.*
- **Dinámicas:** *(unix extension .so, win extension .dll)*
  - Se enlaza al poner el ejecutable en marcha. *(runtime)*

# Ejemplo de creación y uso de bibliotecas

- Partimos de un código *monolítico* (todo-en-uno) donde el programa principal y las funciones que éste emplea están en un único archivo. Llamamos a esta solución: `comp-decomp-a`.
- Se trata de una aplicación que implementa un sencillo algoritmo de compresión de cadenas. Puedes tratar de implementar la descompresión (de ahí el “*decomp*” del nombre).
- Invocado de esta forma: `comp-decomp -c ccccaassssssssssaaaaaaa` produce esta salida: `Compresion de “ccccaassssssssssaaaaaaa”(21) es “4caa8s7a”(8)`.
- Posteriormente crearemos una versión en la que el código del método que *comprime* la cadena y la clase a la cual pertenece, se encuentra en otro proyecto, el cual creará una biblioteca con la que enlazar el programa principal. Llamamos a esta solución: `comp-decomp-b`.

# Comp-decomp-a I

- Se trata de una solución que consta de un solo proyecto.



- El archivo `Program.cs` contiene todo el código de la aplicación, de forma resumida:

# Comp-decomp-a II

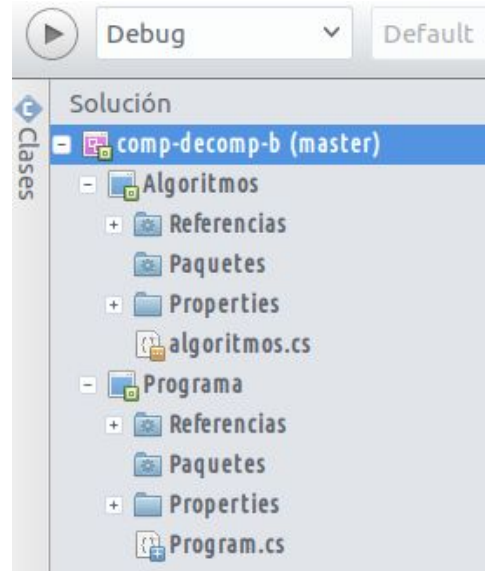
- **Program.cs:**

```
namespace CompDecomp {  
    class Algoritmos {  
        private static int caracteresIguales (string s) {...}  
        public static void comprime (string s, ref string cs) {...}  
    }  
    class MainClass {  
        public static void Main (string[] args) {  
            Console.WriteLine ("Hello compressed World!");  
            string s="ccccaassssssssssaaaaaaa", cs="";  
            Algoritmos.comprime (s, ref cs);  
            Console.WriteLine ("{0}({1}) compressed is [{2}({3})]",  
                               s, s.Length, cs, cs.Length);  
        }  
    }  
}
```



# Comp-decomp-b I

- Se trata de una solución que consta de dos proyectos: Algoritmos y Programa.



- De manera resumida, el contenido de algoritmos.cs y Program.cs es:

# Comp-decomp-b II

- **Program.cs:** *fíjate que pertenece al proyecto "Programa"*

```
namespace CompDecomp {  
    class MainClass {  
        public static void Main (string[] args) {  
            Console.WriteLine ("Hello compressed World!");  
            string s="ccccaassssssssssaaaaaaa", cs="";  
            Algoritmos.comprime (s, ref cs);  
            Console.WriteLine ("{0}({1}) compressed is [{2}({3})]",  
                               s, s.Length, cs, cs.Length);  
        }  
    }  
}
```

# Comp-decomp-b III

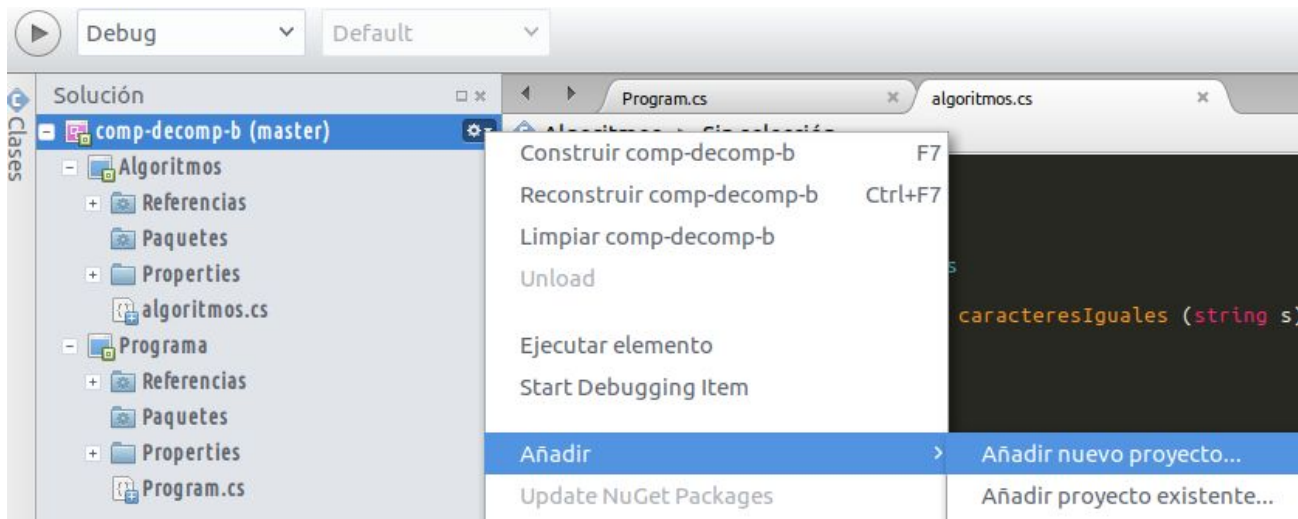
- **algoritmos.cs**: *fíjate que pertenece al proyecto “Algoritmos”*

```
namespace CompDecomp {  
    public class Algoritmos {  
        private static int caracteresIguales (string s) {...}  
        public static void comprime (string s, ref string cs) {...}  
    }  
}
```

- Por cierto...comprueba qué ocurre si quitamos el atributo **public** de la clase Algoritmos.

# Comp-decomp-b IV

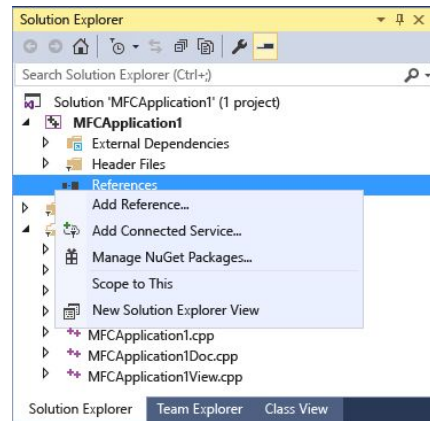
- ¿Cómo se añade un nuevo proyecto a una solución?



- Este botón de configuración, p.e. también nos permite renombrar un proyecto, un archivo, etc...
- También podemos acceder a estas operaciones pulsando el botón derecho del ratón sobre el elemento correspondiente.

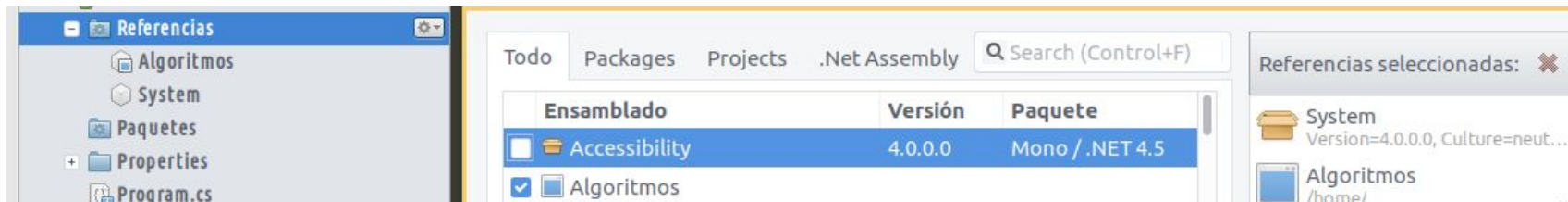
# Comp-decomp-b V

- El proyecto Algoritmos **es de tipo librería y no aplicación de consola.**
- Su finalidad es producir una DLL y no un ejecutable.
- Además de añadir el proyecto a la solución, hemos de añadir una **referencia** a este proyecto (Algoritmos) en el resto de proyectos de la solución que lo empleen, en este ejemplo sólo en el proyecto Programa.
- Seguro que ya te imaginas cómo se hace...



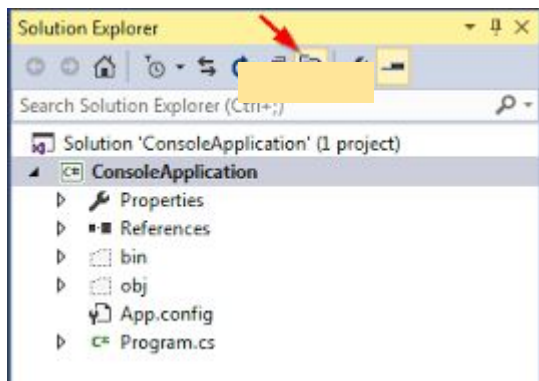
- Lo cual nos mostrará un diálogo como éste donde podremos seleccionarla:

# Comp-decomp-b VI

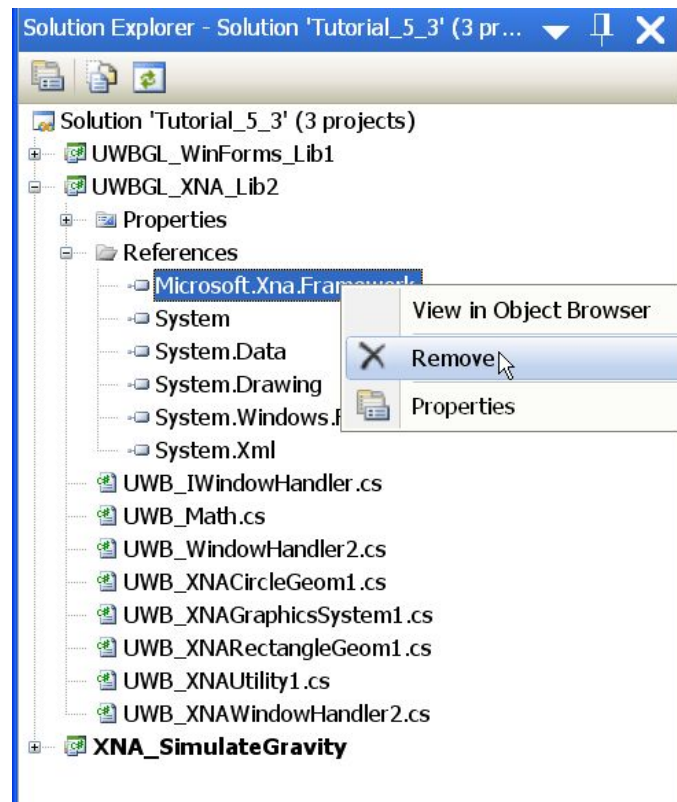


# Carpeta Referencias

- Mira la carpeta **Referencias** de un proyecto:



Qué piensas que son estas referencias?



# Visual Studio y proyectos de tipo “Librería”.

- **Los *proyectos de tipo librería* se pueden crear independientemente** de que tengan un proyecto de tipo “aplicación gráfica o de consola” asociados en la misma solución. De hecho, suele ser lo habitual.
- **Podemos crear nuestros proyectos de tipo *librería* que posteriormente podemos reutilizar en aplicaciones concretas.**
- Es una buena manera de dividir el trabajo dentro de un grupo de programadores. Cada *subgrupo* se dedica a crear una librería.



# En la práctica en grupo...

- Una solución con 2 proyectos:
  - Proyecto Web (con referencia al proyecto de librería)
    - Contendrá la interfaz, manejadores y validaciones
  - Proyecto de Librería de clases con las Capas EN y CAD
    - Carpeta EN
      - Clases EN para entidades del sistema
    - Carpeta CAD
      - Por cada EN una clase CAD para el acceso a datos