Modificadores de acceso

- `public` : No existen restricciones para el acceso a los miembros o tipos que se hayan definido mediante public
- `private` : Sólo son accesibles dentro de la clase en la que se definen
- `internal` : Sólo serán accesibles desde el archivo.cs desde el que se definen. En un archivo.cs se pueden crear varias clases, enum, etc.
- `protected` : Sólo para la clase en la que se usa el protected y las clases que heredan de esta(es como un private para herencias)

Modificadores de miembros y tipos

- `static` : Los miembros declarados con static tienen la capacidad de existir una sola vez, puesto que no se pueden instanciar y por tanto existir mas instancias en memoria
    - El miembro `static` puede utilizarse con clases, campos, métodos, propiedades... Pero no con indizadores, destructores o tipos.
- `const` : El modificador const se utiliza para la definición de constantes
- `event` : Este modificador se utiliza para declarar eventos
- `extern` : El modificador `extern` indica que el método marcado con este modificador se implementa externamente en otro ensamblado. Un uso común del modificador extern es con el atributo DllImport al usar servicios de interoperabilidad para llamar a código no administrado.
- `new` : Este modificador se usa:
    - Para **crear objetos**
    - **Cuando se herede de una clase base** y haya dos métodos de mismo nombre, si quiero usar el método de la clase actual y no el de la clase heredada se usará el modificador **new** en el método declarado.
    - **Como restricción de clase genérica** indicando que el tipo genérico `class<T>` debe tener un constructor público sin parámetros

```
public class clase<T> where T: new() {

}
```

- **Covarianza y Contravarianza** : son propiedades que se atribuyen a jerarquías de clases.
    - `in` :

    - `out` :

- `partial` : Divide la definición de miembros en varios archivos dentro del mismo proyecto. Es muy usado para el desarrollo de **GUIs** con Windows Forms
- `readonly` : Provoca que el campo definido con el modificador readonly **no se pueda modificar**
- `unsafe` : La palabra clave unsafe denota un contexto **no seguro**, que es necesario para cualquier operación que involucre a punteros.

```
unsafe{
    int* x;
}
unsafe void métodoUnsafe(){
    int* x;
}
unsafe class claseUnsafe(){
}
```

- `volatile` : La palabra clave volatile indica que un campo puede ser modificado por varios subprocesos que se ejecutan al mismo tiempo.
- `async` : El modificador **async** se usa para indicar que un método, una expresión lambda o un método anónimo es asíncrono. Un método asincrónico se ejecuta sincrónicamente hasta alcanzar la primera expresión `await`, en la que se suspende el método hasta que se complete la tarea en espera.

Herencia

- `abstract` : El modificador abstract se utiliza principalmente para definir clases base, obligándo a las clases derivadas a implementar los miembros marcados con abstract.
    - Las clases abstractas **no se pueden instanciar**
    - Como las clases abstractas tienden a ser clases base, no se pueden modificar con **sealed**
    - Una clase no abstracta derivada de una clase abstracta debe incluir implementaciones reales de todos los descriptores de acceso y métodos abstractos heredados.
    - Las propiedades abstractas funcionan igual que los métodos abstractos.
- `override` : Se usa para implementar métodos de clases base marcados como **virtual** o **abstract** en las clases derivadas. Cuando se aplica override lo que se hace es operar sobre el método que heredas, por tanto se sobrescribe el método base para agregar el contenido nuevo de la clase derivada
- `virtual` : Al utilizar el modificador virtual sobre un método, propiedad, indizador o declaración de evento, estamos permitiendo que este se pueda sobrescribir en una clase derivad
- `sealed` : Se aplica a **clases, métodos y propiedades**. Los miembros marcados con `sealed` **NO** se puede heredar de ellos.