Liberación de recursos y memoria

Como una buena política de programación siempre es bueno liberar recursos y espacios de memoria como viene explicado aquí.

Using

Una forma segura de liberar siempre los recursos del objeto que se esta usando es utilizar esta instrucción.

La instrucción `using()` solo se puede usar en clases que implementen la interface `IDisposable`. Dispose

```
using(Aes aesAlg = Aes.Create()) {
    //Código usando aesAlg
}
```

Esta instrucción asegura que cuando acabe el bloque de código se usara el `aesAlg.Dispose();`

incluso si hay una excepción en el código.

Creando la liberación de memoria

En la parte anterior con el using liberábamos objetos que ya disponían de un finalizador o destructor, pero ¿y si necesitamos liberar recursos de una clase que hemos desarrollado nosotros mismos?, pues se hará lo siguiente

Dispose

El método Dispose se implementa para liberar recursos de la clase donde se implementa, sobretodo se usa para gestión de código no administrado como usos como conexiones a BBDD, Streams, etc.

1 => Implementando la interface

```
public class clase: IDisposable{
    public void Dispose() {
        this.Dispose(true);
        GC.SuppressFinalize(this);
    }
}
```

`Dispose(true)` llama a un método Dispose sobrecargado que se encargara de liberar los recursos

La llamada al método `SuppressFinalize` informa al recolector de basura (GC) que el objeto se limpió completamente...

2 => Sobrecargando y liberando

```
protected virtual void Dispose(bool disposing) {
    // Preguntamos si Dispose ya fue llamado.
    if (disposing) {
        // Llamamos al Dispose de todos los RECURSOS MANEJADOS.
        this.componentes.Dispose();
        this.dataSetDisposable.Dispose();
    }
        // Finalizamos correctamente los RECURSOS NO MANEJADOS
}
```

Si recibimos `true` llamamos a los Dispose de todos los elementos que hemos usado, por ejemplo conexiones a bbdd, streams, etc.

Tanto si recibimos `true` como `false` llamamos a los Dispose del código NO administrado que tenemos en el programa

---

Finalizadores

Los finalizadores (también denominados destructores) se usan para realizar cualquier limpieza final necesaria cuando el recolector de basura va a liberar el objeto de memoria

- Los finalizadores no se pueden definir en struct. Solo se usan con clases.
- Una clase solo puede tener un finalizador.
- Los finalizadores no se pueden heredar ni sobrecargar.
- No se puede llamar a los finalizadores. Se invocan automáticamente.
- Un finalizador no permite modificadores ni tiene parámetros.

```
class Clase {
    ~Clase() {
        // Instrucciones de limpieza
        /* Se puede poner la llamada al Dispose
         que hemos visto antes*/
        this.Dispose(false);
    }
}
```