

Examen2ª Evaluación

1. (DOSSIER - 1,5 pt) Encontrar cinco errores de normas de estilo en el fichero *loto.cs*, indicando número de línea, error encontrado y solución.

- Nombre de clase debe utilizar estilo PasCal: línea 7. Error *loto*. Sol *LotoDAS2223*.
- Nombre de constantes deben utilizar estilo PasCal: líneas 10 a 12. Errores *MAX_NUMEROS*, *NUMERO_MENOR*, *NUMERO_MAYOR*. Sol *MaximoNumeros*, *NumeroMenor*, *NumeroMayor*.
- Nombre de campos debe utilizar estilo caMel: línea 14. Error *int[] _nums*. Sol *private int[] numerosCombinacion*. Ahora además podemos eliminar el comentario que explica qué es ese campo.
Línea 15. Error *ok*. Sol *combinacionValida*.

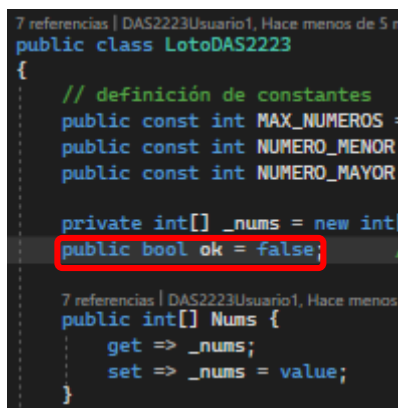
Otros errores:

- Error comentario no existente: línea 23. Error *“//(sin comentario)”*. Sol Eliminar *“//”*.
- Espaciado: línea 27 (entre otras muchas –línea 32, 40,...-): Error *i=0*. Sol *i = 0*;

3. (DOSSIER+COMMIT - 1,5 pt) Si existen, detectar y aplicar al menos tres patrones de refactorización (tanto en el fichero *Loto.cs* como en el fichero *Form1.cs*), indicando el patrón que se aplica y, si es posible aplicarlo con Visual Studio, la opción que se usa.

Clase Loto

Vemos que uno de los campos es público. Es buena idea encapsular los campos para poder restringir su acceso desde fuera. Se validarán el método de su propiedad, que recogerá y dará valor a estos con sus métodos *get* y *set*.



```
7 referencias | DAS2223Usuario1, Hace menos de 5 m
public class LotoDAS2223
{
    // definición de constantes
    public const int MAX_NUMEROS =
    public const int NUMERO_MENOR
    public const int NUMERO_MAYOR

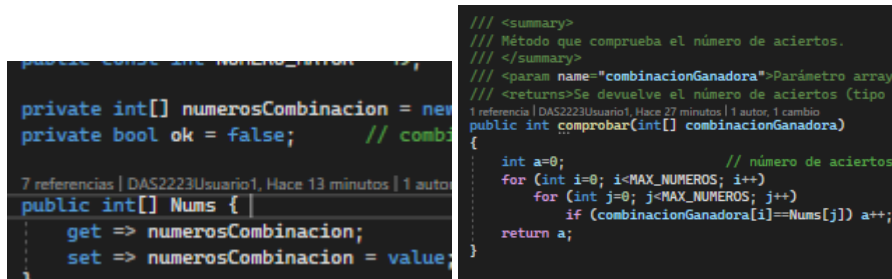
    private int[] _nums = new int[
    public bool ok = false;

    7 referencias | DAS2223Usuario1, Hace menos
    public int[] Nums {
        get => _nums;
        set => _nums = value;
    }
}
```

En VS, si hacemos clic en esa línea, y vamos a *Editar/Refactorizar/Encapsular Campos...* nos lo hace automáticamente.

Renombrar también es una técnica de refactorización, y aquí vamos a utilizarla en la línea 14 para cambiar el nombre de `_nums` a `numerosCombinacion` (campos en caMeI), y línea 87 para cambiar `premi` por `combinaciónGanadora`.

Vamos a editar/Refactorizar/Cambiar nombre y hacemos lo dicho. Resultado:



```

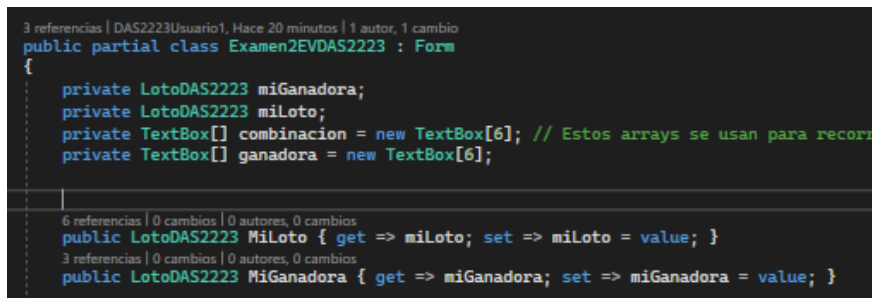
private int[] numerosCombinacion = new int[6];
private bool ok = false; // comb

7 referencias | DAS2223Usuario1, Hace 13 minutos | 1 autor, 1 cambio
public int[] Nums {
    get => numerosCombinacion;
    set => numerosCombinacion = value;
}

1 referencia | DAS2223Usuario1, Hace 27 minutos | 1 autor, 1 cambio
public int comprobar(int[] combinacionGanadora)
{
    int a=0; // número de aciertos
    for (int i=0; i<MAX_NUMEROS; i++)
        for (int j=0; j<MAX_NUMEROS; j++)
            if (combinacionGanadora[i]==Nums[j]) a++;
    return a;
}
    
```

Form1

Encapsularemos los campos `miLoto` y `miGanadora`:



```

3 referencias | DAS2223Usuario1, Hace 20 minutos | 1 autor, 1 cambio
public partial class Examen2EVDAS2223 : Form
{
    private LotoDAS2223 miGanadora;
    private LotoDAS2223 miLoto;
    private TextBox[] combinacion = new TextBox[6]; // Estos arrays se usan para recorrer
    private TextBox[] ganadora = new TextBox[6];

    6 referencias | 0 cambios | 0 autores, 0 cambios
    public LotoDAS2223 MiLoto { get => miLoto; set => miLoto = value; }
    3 referencias | 0 cambios | 0 autores, 0 cambios
    public LotoDAS2223 MiGanadora { get => miGanadora; set => miGanadora = value; }
}
    
```

4. (DOSSIER - 1,5 pt) Realizar el diseño de pruebas (caja negra) para el constructor con parámetro de la clase *loto*.

```

/// <summary>
/// Constructor con Parámetro <paramref name="misnums"/>.
/// </summary>
/// <param name="misnums">Inserta el array de números.
/// </param>
/// <remarks>
/// misnums es un array de enteros con la combinación que quiero crear (
/// </remarks>
2 referencias | DAS2223Usuario1, Hace menos de 5 minutos | 1 autor, 2 cambios
public LotoDAS2223(int[] misnums) // misnumeros: combinación con la que
{
    for (int i=0; i<MAX_NUMEROS; i++) // TODO faltan llaves
    {
        if (misnums[i]>=NUMERO_MENOR && misnums[i]<=NUMERO_MAYOR) {
            int j;
            for (j=0; j<i; j++)
                if (misnums[i]==misnums[j])
                    break;

            if (i==j)
                Nums[i]=misnums[i]; // validamos la combinación
            else {
                Ok=false;
                return;
            }
        }
        else
        {
            Ok=false; // La combinación no es válida, terminamos
            return;
        }
    }
    Ok=true;
}

```

En el constructor con parámetro se introduce como parámetro, un vector que incluye los números con los que jugamos. En dicho constructor se validará si dicha combinación es o no válida para jugar. Sabemos que cada elemento entero del vector (cada número jugador), ha de estar entre 1 y 49 (ambos inclusive) para que sean válidos. Jugarán 6 números y no pueden repetirse.

Comenzamos por definir las distintas clases de equivalencia:

- PR1: Número < 1. (ERROR NÚMERO FUERA DE RANGO). Val. Límite -2.
- PR2: Número >= 1 && Número <= 49. (VÁLIDO). Val. Límite 1, 49.
- PR3: Número > 49. (ERROR NÚMERO FUERA DE RANGO). Val. Límite 50.

Casos de prueba:

- PR1: CPrueba -2, -50. Salida Esperada (ERROR NÚMERO FUERA DE RANGO)
- PR2: CPrueba 1, 30, 49. Salida Esperada (Comb Valida true - VÁLIDO)
- PR3: CPrueba 50, 100. Salida Esperada (ERROR NÚMERO FUERA DE RANGO)