

El primer error está en el nombre de la clase. Los nombres de las clases y constructores deben seguir la notación PasCal.

```
namespace LotoClassNS
{
    // Clase que almacena una combinación de la lotería
    //
    public class Loto
    {
```

Solución: cambiar el nombre de la clase y los constructores por LotoRIA2223.

El segundo error está en el nombre de los atributos. Los atributos deben tener nombres que sean autodescriptivos y a poder ser en notación caMel.

```
private int[] _nums = new int[MAX_NUMEROS]; // num
public bool ok = false; // combinación válida

public int[] Nums {
    get => _nums;
    set => _nums = value;
}
```

Solución: cambiamos los atributos por “numero” y “combinacionValida” en todos los lugares donde aparezcan.

```
private int[] numeros = new int[MAX_NUMEROS]; // numeros de la combinación
public bool combinacionValida = false; // combinación válida (si es aleatoria,

public int[] Numeros {
    get => numeros;
    set => numeros = value;
}
```

En el constructor que setea números aleatorios encontramos otro error. En la declaración de variables, el iterador i del bloque do ... while está bien definido. Sin embargo, la j debería definirse en el bucle for.

```

public LotoRIA2223()
{
    Random r = new Random();    // clase generadora de

    int i=0, j, num;

    do                // generamos la combinación
    {
        num = r.Next(NUMERO_MENOR, NUMERO_MAYOR + 1);
        for (j=0; j<i; j++)    // comprobamos que el nú
            if (Nums[j]==num)
                break;
        if (i==j)            // Si i==j, el número n
        {
            Nums[i]=num;
            i++;
        }
    } while (i<MAX_NUMEROS);

    combinacionValida=true;
}

```

Solución: declaramos la j en el for.

En el constructor con parámetro encontramos otro error de formato. El nombre del parámetro debe estar en formato camel

```

public LotoRIA2223(int[] misnums) // misnumeros: combinación con la que queremos inicializar la clase
{
    for (int i = 0; i < MAX_NUMEROS; i++)
        if (misnums[i] >= NUMERO_MENOR && misnums[i] <= NUMERO_MAYOR) {
            int j;
            for (j=0; j<i; j++)
                if (misnums[i] == Numeros[j])
                    break;
            if (i == j)
                Numeros[i] = misnums[i]; // validamos la combinación
            else {
                combinacionValida = false;
                return;
            }
        }
    else
    {
        combinacionValida = false;    // La combinación no es válida, terminamos
        return;
    }
    combinacionValida = true;
}

```

Solución: misnums pasa a llamarse misNumeros

Patrones de refactorización

Si miramos el código, podemos ver que los símbolos y comparadores no se leen correctamente porque no están espaciados con las variables, de modo que arreglaremos eso también. Queda de este modo para el caso del primer constructor, pero lo aplicaremos a todo

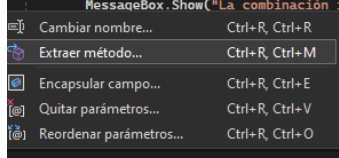
```
// En el caso de que el constructor sea vacío, se genera una
//
public LotoRIA2223()
{
    Random r = new Random();    // clase generadora de números
    int i = 0, num;

    do                // generamos la combinación
    {
        num = r.Next(NUMERO_MENOR, NUMERO_MAYOR + 1);    //
        for (int j = 0; j < i; j++)    // comprobamos que el
            if (Nums[j] == num)
                break;
        if (i == j)                // Si i==j, el número no s
        {
            Nums[i] = num;
            i++;
        }
    } while (i < MAX_NUMEROS);

    combinacionValida = true;
}
```

En el formulario de la solución podemos aplicar también una extracción de método con la herramienta disponible en visual en el código destacado en la imagen:

```
if (miLoto.ok)
{
    nums = new int[6];
    for (int i = 0; i < 6; i++)
        nums[i] = Convert.ToInt32(combinacion[i].Text);
    int aciertos = miGanadora.comprobar(nums);
    if (aciertos < 3)
        MessageBox.Show("No ha resultado premiada");
    else
        MessageBox.Show("¡Enhorabuena! Tiene una combinación con " + Convert.ToString(aciertos) + " aciertos");
}
else
    MessageBox.Show("La combinación introducida no es válida");
```



Queda de la siguiente manera:

```

private void btComprobar_Click(object sender, EventArgs e)
{
    int[] nums = new int[6];
    for (int i = 0; i < 6; i++)
        nums[i] = Convert.ToInt32(combinacion[i].Text);
    miLoto = new Loto(nums);
    if (miLoto.ok)
    {
        nums = comprobarLoto();
    }
    else
        MessageBox.Show("La combinación introducida no es válida");
}

private int[] comprobarLoto()
{
    int[] nums = new int[6];
    for (int i = 0; i < 6; i++)
        nums[i] = Convert.ToInt32(combinacion[i].Text);
    int aciertos = miGanadora.comprobar(nums);
    if (aciertos < 3)
        MessageBox.Show("No ha resultado premiada");
    else
        MessageBox.Show("Enhorabuena! Tiene una combinación con " + Convert.ToString(aciertos) + " aciertos");
    return nums;
}

```

Otra técnica de refactorización es el encapsulamiento de campos. En el mismo archivo del form, podemos convertir en privadas las variables `miLoto` y `miGanadora`, ya que no debemos permitir que se acceda desde el exterior a dichas variables.