

## Seminario de Lenguajes (.NET)

### Práctica 8

**Ejercicios obligatorios para el coloquio:** Los **ejercicios 5, 6, 10 y 11** de esta práctica forman parte del conjunto de ejercicios de programación obligatorios que el alumno debe resolver y exponer de manera oral sobre máquina el día del coloquio hacia final de la cursada.

1) Declarar los tipos delegados necesarios para que el siguiente programa compile y produzca la salida en la consola indicada

```
class Program
{
    static void Main(string[] args)
    {
        Del1 d1 = delegate (int x) { Console.WriteLine(x); };
        d1(10);

        Del2 d2 = x => Console.WriteLine(x.Length);
        d2(new int[] { 2, 4, 6, 8 });

        Del3 d3 = x =>
        {
            int sum = 0;
            for (int i = 1; i <= x; i++)
            {
                sum += i;
            }
            return sum;
        };
        Console.WriteLine(d3(10));

        Del4 d4 = new Del4(LongitudPar);
        Console.WriteLine(d4("hola mundo"));
    }
    public static bool LongitudPar(string st)
    {
        return st.Length % 2 == 0;
    }
}
```

Salida por  
consola

```
10
4
55
True
```

2) Responder sobre el siguiente código

```
delegate void AccionInt(ref int i);
class Program
{
    static void Main(string[] args)
    {
        AccionInt a1 = (ref int i) => i = i * 2;
        a1 += a1;
        a1 += a1;
        a1 += a1;
        int i = 1;
        a1(ref i);
        // Responder respecto de este punto en el programa
    }
}
```

¿Cuál es el tamaño de la lista de invocación de **a1** y cual es el valor de la variable **i** luego de la invocación **a1(ref i)**?

3) ¿Qué obtiene un método anónimo (o expresión lambda) cuando accede a una variable definida en el entorno que lo rodea, una copia del valor de la variable o la referencia a dicha variable? Tip: Observar la salida por consola del siguiente código:

```
static void Main(string[] args)
{
    int i = 10;
    Action a = delegate ()
    {
        Console.WriteLine(i);
    };
    a.Invoke();
    i = 20;
    a.Invoke();
}
```

- 4) Teniendo en cuenta lo respondido en el ejercicio anterior, ¿Qué salida produce en la consola la ejecución del siguiente método?

```
static void Main(string[] args)
{
    Action[] acciones = new Action[10];
    for (int i = 0; i < 10; i++)
    {
        acciones[i] = () => Console.WriteLine(i + " ");
    }
    foreach (var a in acciones)
    {
        a.Invoke();
    }
}
```

- 5) En este ejercicio, se requiere extender el tipo `int[]` con algunos métodos de extensión. Se presenta el código del método de extensión `Print(this int[] vector, string leyenda)` que imprime en la consola los elementos del vector precedidos por una leyenda que se pasa como parámetro. Se requiere codificar el método de extensión `Select(...)` que recibe como parámetro un delegado de tipo `FuncionEntera` y devuelve un nuevo vector de enteros producto de aplicar la función recibida como parámetro a cada uno de los elementos del vector. El siguiente programa debe producir la salida indicada.

```
delegate int FuncionEntera(int n);
class Program
{
    static void Main(string[] args)
    {
        int[] vector = new int[] { 1, 2, 3, 4, 5 };
        vector.Print("Valores iniciales: ");
        var vector2 = vector.Select(n => n * 3);
        vector2.Print("Valores triplicados: ");
        vector.Select(n => n * n).Print("Cuadrados: ");
    }
}
```

**Salida por consola**

```
Valores iniciales: 1, 2, 3, 4, 5
Valores triplicados: 3, 6, 9, 12, 15
Cuadrados: 1, 4, 9, 16, 25
```

Para ello, completar el código de la siguiente clase estática `VectorDeEnterosExtension`

```

static class VectorDeEnterosExtension
{
    public static void Print(this int[] vector, string leyenda)
    {
        string st = leyenda;
        if (vector.Length > 0)
        {
            foreach (int n in vector) st += n + ", ";
            st = st.Substring(0, st.Length - 2);
        }
        Console.WriteLine(st);
    }
    public static int[] Select(. . . )
    {
        . . .
    }
}

```

6) Agregar al ejercicio anterior el método de extensión `Where(...)` para el tipo `int[]` que recibe como parámetro un delegado de tipo `Predicado` y devuelve un nuevo vector de enteros con los elementos del vector que cumplen ese predicado. El siguiente programa debe producir la salida indicada.

```

delegate bool Predicado(int n);
delegate int FuncionEntera(int n);
class Program
{
    static void Main(string[] args)
    {
        int[] vector = new int[] { 1, 2, 3, 4, 5 };
        vector.Print("Valores iniciales: ");
        vector.Where(n => n % 2 == 0).Print("Pares: ");
        vector.Where(n => n % 2 == 1).Select(n => n * n).Print("Impares al cuadrado: ");
    }
}

```

Salida por  
consola

```

Valores iniciales: 1, 2, 3, 4, 5
Pares: 2, 4
Impares al cuadrado: 1, 9, 25

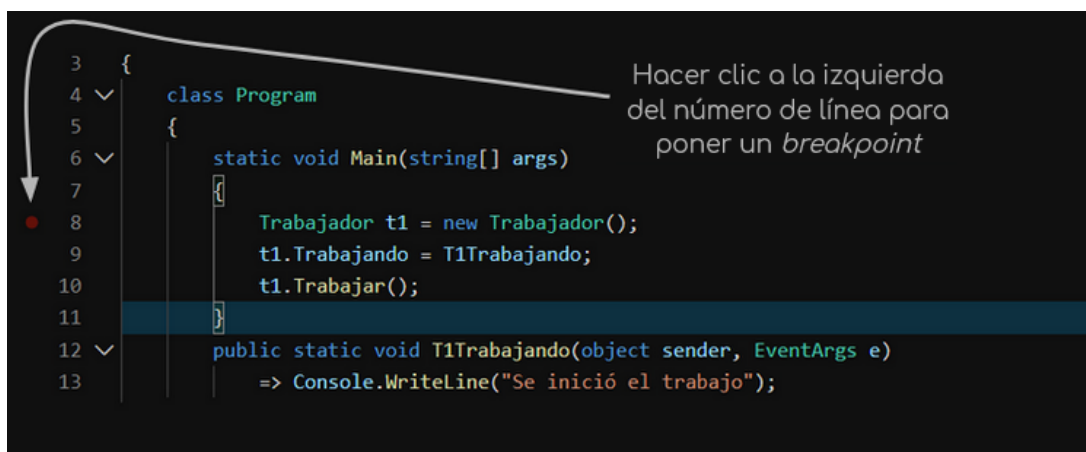
```

7) Dado el siguiente código:

```
class Program
{
    static void Main(string[] args)
    {
        Trabajador t1 = new Trabajador();
        t1.Trabajando = T1Trabajando;
        t1.Trabajar();
    }
    public static void T1Trabajando(object sender, EventArgs e)
        => Console.WriteLine("Se inició el trabajo");
}
class Trabajador
{
    public EventHandler Trabajando; //No es necesario definir un tipo delegado propio
                                    //porque la plataforma provee el tipo EventHandler
                                    //que se adecua a lo que se necesita

    public void Trabajar()
    {
        Trabajando(this, EventArgs.Empty);
        //realiza algún trabajo
        Console.WriteLine("Trabajo concluido");
    }
}
```

a) Ejecutar paso a paso el programa y observar cuidadosamente su funcionamiento. Para ejecutar paso a paso colocar un punto de interrupción (*breakpoint*) en la primera línea ejecutable del método **Main()**



Ejecutar el programa y una vez interrumpido, proseguir paso a paso, en general la tecla asociada para ejecutar paso a paso entrando en los métodos que se invocan es F11, sin embargo también es posible utilizar el botón de la barra que aparece en la parte superior del editor cuando el programa está en ejecución.

```
Program.cs X
Program.cs > {} ej4 > ej4.Program > Main(string[] args)
1 using System;
2 namespace ej4
3 {
4     class Program
5     {
6         static void Main(string[] args)
7         {
8             Trabajador t1 = new Trabajador();
9             t1.Trabajando = T1Trabajando;
10            t1.Trabajar();
11        }
    }
}
```

Una vez interrumpido, hacer clic acá para ir paso a paso entrando en los métodos que se invocan

- b) ¿Qué salida produce por Consola?
- c) Borrar (o comentar) la instrucción `t1.Trabajando = T1Trabajando;` del método `Main` y contestar:
  - c.1) ¿Cuál es el error que ocurre? ¿Dónde y por qué?
  - c.2) ¿Cómo se debería implementar el método `Trabajar()` para evitarlo? Resolverlo.
- d) Eliminar el método `T1Trabajando` de la clase `Program` y suscribirse al evento con una expresión lambda.
- e) Reemplazar el campo público `Trabajando` de la clase `Trabajador`, por un evento público generado por el compilador (event notación abreviada). ¿Qué operador se debe usar en la suscripción?
- f) Cambiar en la clase `Trabajador` el evento generado automáticamente por uno implementado de manera explícita con los dos descriptores de acceso y haciendo que, al momento en que alguien se suscriba al evento, se dispare el método `Trabajar()`, haciendo innecesaria la invocación `t1.Trabajar();` en la clase `Program`.

8) Analizar el siguiente código:

```
class Program
{
    static void Main()
    {
        ContadorDeLineas contador = new ContadorDeLineas();
        contador.Contar();
    }
}
class ContadorDeLineas
{
    private int _cantLineas = 0;
    public void Contar()
    {
        Ingresador _ingresador = new Ingresador();
        _ingresador.Contador = this;
        _ingresador.Ingresar();
        Console.WriteLine($"Cantidad de líneas ingresadas: {_cantLineas}");
    }
    public void UnaLineaMas() => _cantLineas++;
}
class Ingresador
{
    public ContadorDeLineas Contador {get;set;}
    public void Ingresar()
    {
        string st = Console.ReadLine();
        while (st != "")
        {
            Contador.UnaLineaMas();
            st = Console.ReadLine();
        }
    }
}
```

Existe un alto nivel de acoplamiento entre las clases **ContadorDeLineas** e **Ingresador**, habiendo una referencia circular: un objeto **ContadorDeLineas** posee una referencia a un objeto **Ingresador** y éste último posee una referencia al primero. Esto no es deseable, hace que el código sea difícil de mantener. Eliminar esta referencia circular utilizando un evento, de tal forma que **ContadorDeLineas** posea una referencia a **Ingresador** pero que no ocurra lo contrario.

9) Codificar una clase **Ingresador** con un método público **Ingresar()** que permita al usuario ingresar líneas por la consola hasta que se ingrese la línea con la palabra **"fin"**. **Ingresador** debe implementar dos eventos. Uno sirve para notificar que se ha ingresado una línea vacía ( **" "** ). El otro para indicar que se ha ingresado un valor numérico (debe comunicar el valor del número ingresado como argumento cuando se genera el evento). A modo de ejemplo observar el siguiente código que hace uso de un objeto **Ingresador**.

```

. . .
Ingresador ingresador = new Ingresador();
ingresador.LineaVacíaIngresada += (sender, e) =>
    { Console.WriteLine("Se ingresó una línea en blanco"); };
ingresador.NroIngresado += (sender, e) =>
    { Console.WriteLine($"Se ingresó el número {e.Valor}"); };
ingresador.Ingresar();
. . .

```

10) Codificar la clase **Temporizador** con un evento **Tic** que se genera cada cierto intervalo de tiempo medido en milisegundos una vez que el temporizador se haya habilitado. La clase debe contar con dos propiedades: **Intervalo** de tipo **int** y **Habilitado** de tipo **bool**. No se debe permitir establecer la propiedad **Habilitado** en **true** si no existe ninguna suscripción al evento **Tic**. No se debe permitir establecer el valor de **Intervalo** menor a 100. En el lanzamiento del evento, el temporizador debe informar la cantidad de veces que se provocó el evento. Para detener los eventos debe establecerse la propiedad **Habilitado** en **false**. A modo de ejemplo, el siguiente código debe producir la salida indicada.

```

public static void Ejecutar()
{
    Temporizador t = new Temporizador();
    t.Tic += (sender, e) =>
    {
        Console.Write(DateTime.Now.ToString("HH:mm:ss") + " ");
        if (e.Tics == 5)
        {
            t.Habilitado = false;
        }
    };
    t.Intervalo = 2000;
    t.Habilitado = true;
    Console.WriteLine();
    t.Intervalo = 1000;
    t.Habilitado = true;
}

```

Salida por  
consola

```

10:58:43 10:58:45 10:58:47 10:58:49 10:58:51
10:58:52 10:58:53 10:58:54 10:58:55 10:58:56

```

**Nota:** La consola interna del Visual Studio Code puede que no se refresque cada vez que se produce un Tic, por lo tanto la ejecución del programa se verá mejor en una terminal externa.



11) Definir el delegado `PrecioCambiadoEventHandler`, la clase `PrecioCambiadoEventArgs` y la clase `Articulo` de tal forma que el código siguiente produzca la salida por consola que se indica

```
class Program
{
    static void Main()
    {
        Articulo a = new Articulo();
        a.PrecioCambiado += precioCambiado;
        a.Codigo = 1;
        a.Precio = 10;
        a.Precio = 12;
        a.Precio = 12;
        a.Precio = 14;
    }
    public static void precioCambiado(object sender, PrecioCambiadoEventArgs e)
    {
        string texto = $"Artículo {e.Codigo} valía {e.PrecioAnterior}";
        texto += $" y ahora vale {e.PrecioNuevo}";
        Console.WriteLine(texto);
    }
}
```

**Salida por  
consola**

```
Artículo 1 valía 0 y ahora vale 10
Artículo 1 valía 10 y ahora vale 12
Artículo 1 valía 12 y ahora vale 14
```

La clase `Articulo` cuenta con las propiedades de lectura/escritura `Codigo` y `Precio`. Además posee el evento `PrecioCambiado` que se produce cuando se cambia el valor de la propiedad `Precio` (observar que si se asigna el mismo valor el evento no se produce).