Seminario de Lenguajes (.NET)

Práctica correspondiente a las teorías 7 y 8

INTERFACES

1) Codificar las clases e interfaces necesarias para modelar un sistema que trabaja con las siguientes entidades: Autos, Libros, Películas, Personas y Perros. Algunas de estas entidades pueden ser: alquilables (se pueden alquilar a una persona y ser devueltas por una persona), vendibles (se pueden vender a una persona), lavables (se pueden lavar y secar) reciclables (se pueden reciclar) y atendibles (se pueden atender). A continuación se describen estas relaciones:

- Son Alquilables: Libros y Películas
- Son Vendibles: Autos y Perros
- Son Lavables: Autos
- Son Reciclables: Libros y Autos
- Son Atendibles: Personas y Perros

Completar el código de la clase estática Procesador:

```
static class Procesador
{
    public static void Alquilar(IAlquilable x, Persona p) => x.SeAlquilaA(p);
    public static . . .
    . . .
}
```

La ejecución del siguiente código debe mostrar en la consola la salida indicada:

```
Auto auto = new Auto();
Libro libro = new Libro();
Persona persona = new Persona();
Perro perro = new Perro();
Pelicula pelicula = new Pelicula();
Procesador.Alquilar(pelicula, persona);
Procesador.Alquilar(libro, persona);
Procesador.Atender(persona);
Procesador.Atender(perro);
Procesador.Devolever(pelicula, persona);
Procesador.Devolever(libro, persona);
Procesador.Lavar(auto);
Procesador.Reciclar(libro);
Procesador.Reciclar(auto);
Procesador.Secar(auto);
Procesador.Vender(auto, persona);
Procesador.Vender(perro, persona);
```

Salida por consola

```
Alquilando película a persona
Alquilando libro a persona
Atendiendo persona
Atendiendo perro
Película devuelta por persona
Libro devuelto por persona
Lavando auto
Reciclando libro
Reciclando auto
Secando auto
Vendiendo auto a persona
Vendiendo perro a persona
```

2) Incorporar al ejercicio anterior la posibilidad también de lavar a los perros. También se debe incorporar una clase derivada de Película, las "películas clásicas" que además de alquilarse pueden venderse. Estos cambios deben poder realizarse sin necesidad de modificar la clase estática Procesador. El siguiente código debe producir la salida indicada:

```
Auto auto = new Auto();
Libro libro = new Libro();
Persona persona = new Persona();
Perro perro = new Perro();
Pelicula pelicula = new Pelicula();
Procesador.Alquilar(pelicula, persona);
Procesador.Alquilar(libro, persona);
Procesador.Atender(persona);
Procesador.Atender(perro);
Procesador.Devolever(pelicula, persona);
Procesador.Devolever(libro, persona);
Procesador.Lavar(auto);
Procesador.Reciclar(libro);
Procesador.Reciclar(auto);
Procesador.Secar(auto);
Procesador.Vender(auto, persona);
Procesador.Vender(perro, persona);
Procesador.Lavar(perro);
Procesador.Secar(perro);
PeliculaClasica peliculaClasica = new PeliculaClasica();
Procesador.Alquilar(peliculaClasica, persona);
Procesador.Devolever(peliculaClasica, persona);
Procesador.Vender(peliculaClasica, persona);
```

Salida por consola

```
Alquilando película a persona
Alquilando libro a persona
Atendiendo persona
Atendiendo perro
Película devuelta por persona
Libro devuelto por persona
Lavando auto
Reciclando libro
Reciclando auto
Secando auto
Vendiendo auto a persona
Vendiendo perro a persona
Lavando perro
Secando perro
Alquilando película clásica a persona
Película clásica devuelta por persona
Vendiendo película clásica a persona
```

3) Incorporar al ejercicio anterior las interfaces y métodos necesarios para que el siguiente código produzca la salida indicada:

```
var lista = new List<object>() {
          new Persona(),
          new Auto()
     };
foreach (IComercial c in lista)
{
   c.Importa();
}
foreach (IImportante i in lista)
{
   i.Importa();
}
(lista[0] as Persona)?.Importa();
(lista[1] as Auto)?.Importa();
```

Salida por consola

```
Persona vendiendo al exterior
Auto que se vende al exterior
Persona importante
Auto importante
Método Importar() de la clase Persona
Método Importar() de la clase Auto
```

4) Incorporar al ejercicio anterior las interfaces, propiedades y métodos necesarios para que el siguiente código produzca la salida indicada:

```
var vector = new INombrable[] {
     new Persona() {Nombre="Zulema"},
     new Perro() {Nombre="Sultán"},
     new Persona() {Nombre="Claudia"},
     new Persona() {Nombre="Carlos"},
     new Perro() {Nombre="Chopper"},
};
Array.Sort(vector);  //debe ordenar por Nombre alfabéticamente
foreach (INombrable n in vector)
{
    Console.WriteLine($"{n.Nombre}: {n}");
}
```

Salida por consola

```
Carlos: Carlos es una persona
Chopper: Chopper es un perro
Claudia: Claudia es una persona
Sultán: Sultán es un perro
Zulema: Zulema es una persona
```

5) Modificar el ejercicio anterior para que el siguiente código produzca la salida indicada:

```
var vector = new INombrable[] {
     new Persona() {Nombre="Zulema"},
     new Perro() {Nombre="Sultán"},
     new Persona() {Nombre="Claudia"},
     new Persona() {Nombre="Carlos"},
     new Perro() {Nombre="Chopper"},
};
Array.Sort(vector);  //debe ordenar por Nombre alfabéticamente
foreach (INombrable n in vector)
{
    Console.WriteLine($"{n.Nombre}: {n}");
}
```

Salida por consola

```
Carlos: Carlos es una persona
Claudia: Claudia es una persona
Zulema: Zulema es una persona
Chopper: Chopper es un perro
Sultán: Sultán es un perro
```

Es decir, el ordenamiento ahora da prioridad a las personas sobre los perros, primero se listan las personas, ordenadas alfabéticamente, y luego los perros, también ordenados alfabéticamente.
Tip: Sólo es necesario cambiar la forma en que un perro o una persona se sabe comparar contra otro objeto.

6) Modificar el ejercicio anterior para que el siguiente código produzca la salida indicada:

```
var vector = new INombrable[]  {
       new Persona() {Nombre="Ana María"},
       new Perro() {Nombre="Sultán"},
       new Persona() {Nombre="Ana"},
       new Persona() {Nombre="José Carlos"},
       new Perro() {Nombre="Chopper"}
};
Array.Sort(vector, new ComparadorLongitudNombre());//ordena por longitud de Nombre
foreach (INnombrable n in vector)
{
     Console.WriteLine($"{n.Nombre.Length}: {n.Nombre}");
}
```

Salida por consola

```
3: Ana
6: Sultán
7: Chopper
9: Ana María
11: José Carlos
```

7) Proponer una forma para conseguir un ordenamiento aleatorio de todos los elementos de un vector de objetos (**`object[]`**) utilizando **`Array.Sort().`** Cada vez que se invoque debe producir un ordenamiento aleatorio diferente. (Investigar la clase **`System.Random`**)

8) Codificar usando iteradores los métodos:
`Rango(i, j, p)` que devuelve la secuencia de enteros desde **`i`** hasta **`j`** con un paso de **`p`**
`Potencia(b,k)` que devuelve la secuencia $b^1, b^2, ... b^k$
`DivisiblePor(e,i)` retorna los elementos de **`e`** que son divisibles por **`i`**
Observar la salida que debe producir el siguiente código:

```
using System.Collections;

IEnumerable rango = Rango(6, 30, 3);
IEnumerable potencias = Potencias(2, 10);
IEnumerable divisibles = DivisiblesPor(rango, 6);
foreach (int i in rango)
{
    Console.Write(i + " ");
}
Console.WriteLine();
foreach (int i in potencias)
{
    Console.Write(i + " ");
}
Console.WriteLine();
foreach (int i in divisibles)
{
    Console.Write(i + " ");
}
Console.WriteLine();
```

Salida por consola

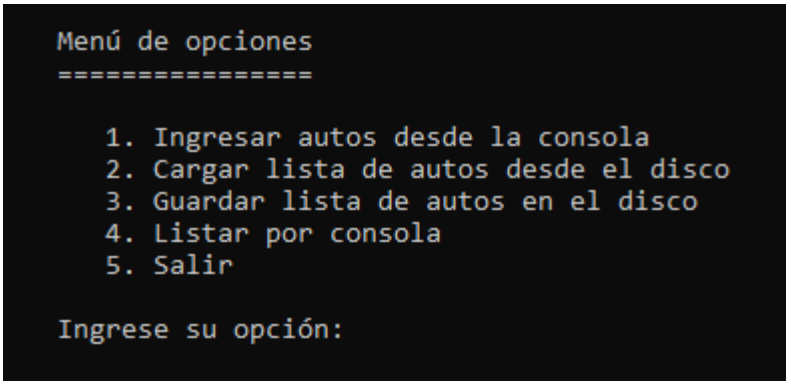
```
6 9 12 15 18 21 24 27 30
2 4 8 16 32 64 128 256 512 1024
6 12 18 24 30
```

9) Codificar un programa que permita al usuario escribir un texto por consola. El mismo puede constar de varios párrafos. Se considera el fin de la entrada cuando el usuario ingresa una línea vacía, en ese momento el programa solicitará al usuario el nombre del archivo para guardar el texto escrito. Si el usuario escribe un nombre de archivo válido, se guarda el texto ingresado en ese archivo, de lo contrario no se hace nada y termina el programa.

a) Utilizando la instrucción using

b) Sin utilizar la instrucción using

10) Codificar una aplicación para manejar una lista de autos (dos atributos: Marca y Modelo) con el siguiente menú de opciones por consola:



La opción 1 permite al usuario agregar autos a la lista actual en memoria ingresando la marca y el modelo por la consola. El final de la entrada se detecta al ingresar una marca vacía (sin caracteres).
La opción 2, carga en memoria una lista de autos previamente guardada en algún archivo de texto.
La opción 3 guarda en un archivo de texto la lista actual en memoria. Se puede implementar de infinidad de maneras, por ejemplo se podría guardar cada auto en dos líneas consecutivas, la primera para la marca y la segunda para el modelo.
La opción 4 produce un listado por consola de todos los autos en la lista actual en memoria.

Tip: La interacción con el menú puede resolverse de la siguiente manera:

```
   . . .
   ConsoleKeyInfo tecla;
   do
   {
      tecla = Console.ReadKey(true);
      switch (tecla.KeyChar)
      {
         case '1': . . .
         case '2': . . .
         case '3': . . .
         case '4': . . .
      }
   } while (tecla.KeyChar != '5');
}
```

DELEGADOS

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

```
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;
};
int resultado = d3(10);
Console.WriteLine(resultado);

Del4 d4 = new Del4(LongitudPar);
Console.WriteLine(d4("hola mundo"));


bool LongitudPar(string st)
{
   return st.Length % 2 == 0;
}
```

Salida por consola

```
10
4
55
True
```

2) ¿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:

```
int i = 10;
Action a = delegate ()
{
    Console.WriteLine(i);
};
a.Invoke();
i = 20;
a.Invoke();
```

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

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

4) 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 **`Seleccionar(...)`** 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.

```
-------Program.cs---------
int[] vector = new int[] { 1, 2, 3, 4, 5 };
vector.Print("Valores iniciales: ");
var vector2 = vector.Seleccionar(n => n * 3);
vector2.Print("Valores triplicados: ");
vector.Seleccionar(n => n * n).Print("Cuadrados: ");

-------FuncionEntera.cs---------
delegate int FuncionEntera(int n);
```

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[] Seleccionar(. . . )
    {
        . . .
    }
}
```

5) Agregar al ejercicio anterior el método de extensión **`Donde(...)`** 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.

```
-------Program.cs---------
int[] vector = new int[] { 1, 2, 3, 4, 5 };
vector.Print("Valores iniciales: ");
vector.Donde(n => n % 2 == 0).Print("Pares: ");
vector.Donde(n => n % 2 == 1).Seleccionar(n => n * n).Print("Impares al cuadrado: ");

-------Predicado.cs---------
delegate bool Predicado(int n);

-------FuncionEntera.cs---------
delegate int FuncionEntera(int n);
```

Salida por consola

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

EJERCICIOS COMPLEMENTARIO

Estos ejercicios se corresponden con el material complementario presentado al final de la teoría 8.

1) Responder sobre el siguiente código

```
----------Program.cs----------
AccionInt a1 = (ref int i) => i = i * 2;
a1 += a1;
a1 += a1;
a1 += a1;
int i = 1;
a1(ref i);
----------AccionInt.cs----------
delegate void AccionInt(ref int i);
```

¿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)`**?

2) Dado el siguiente código:

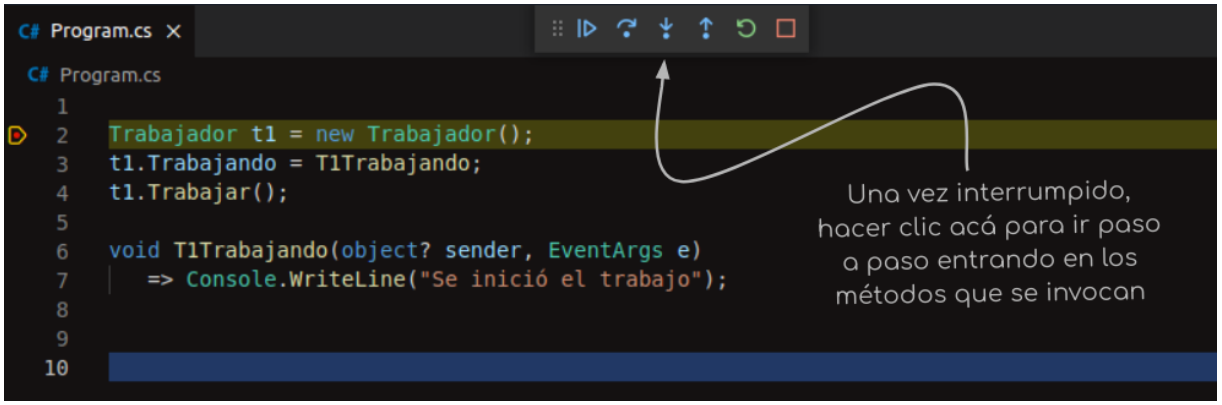
```
-------Program.cs---------
Trabajador t1 = new Trabajador();
t1.Trabajando = T1Trabajando;
t1.Trabajar();

void T1Trabajando(object? sender, EventArgs e)
  => Console.WriteLine("Se inició el trabajo");

-------Trabajador.cs---------
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 con la ejecución interrumpida.



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` en `Program.cs` 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 `Program.cs`

3) Analizar el siguiente código:

```
-------Program.cs---------
ContadorDeLineas contador = new ContadorDeLineas();
contador.Contar();
```

```
-------ContadorDeLineas.cs---------
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++;
}

-------Ingresador.cs---------
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.

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

5) 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.

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

Salida por consola

```
14:20:50
14:20:52
14:20:54
14:20:56
14:20:58
```