Seminario de Lenguajes (.NET)

Práctica 7

Tema: Delegados

Importante: El ejercicio 15 de esta práctica debe entregarse para su evaluación. La fecha de entrega se establecerá más adelante durante el transcurso de la cursada.

1) Cree una nueva solución en el SharpDevelop y codifique el siguiente programa:

```
using System;
delegate void TrabajandoEventHandler();
class Program
{
	public static void Main()
	{
		Trabajador o=new Trabajador();
		o.Trabajando = new TrabajandoEventHandler(F);
		Console.WriteLine("Todo listo para comenzar");
		o.Trabajar();
		Console.ReadKey(true);
	}
	private static void F(){
		Console.WriteLine("El trabajo se ha iniciado");
	}
}
class Trabajador{
    public TrabajandoEventHandler Trabajando;
      public void Trabajar(){
		Trabajando();
		//realiza algún trabajo
		Console.WriteLine("Trabajo concluido");
	}
}
```

a) Ejecute paso a paso el programa y observe cuidadosamente su funcionamiento. Para ejecutar paso a paso coloque un punto de interrupción (breakpoint) en la primera línea ejecutable del método `Main()`, ejecute el programa y una vez interrumpido, prosiga paso a paso presinando la tecla F11.

b) ¿Qué salida produce por Consola?

2) Borre (o comente) la segunda la instrucción del método Main()

```
o.Trabajando = new TrabajandoEventHandler(F);
```

Ejecute el programa nuevamente y conteste:

a) ¿Qué sucede?

b) ¿En qué instrucción se produce el error?

c) ¿A qué se debe la ocurrencia del error?

d) ¿Si en lugar de utilizar la instrucción "`Trabajando();`" en el método `Trabajar()` se utiliza "`if (Trabajando != null) Trabajando();`" se resuelve el problema?

Observe que un objeto que posea delegados no puede asegurar que "quienes lo utilicen" van a asignarlo en algún momento, por lo tanto al tratar de invocarlo es posible que el delegado posee el valor null.

3) Modifique la clase Program de la siguiente manera (se ha sombreado las líneas de código agregadas)

```
class Program
{
    public static void Main()
    {
        Trabajador o=new Trabajador();
        o.Trabajando = new TrabajandoEventHandler(F);
        o.Trabajando += new TrabajandoEventHandler(G);
        Console.WriteLine("Todo listo para comenzar");
        o.Trabajar();
        Console.ReadKey(true);
    }
    private static void F(){
        Console.WriteLine("El trabajo se ha iniciado");
    }
    private static void G(){
        Console.WriteLine("Ejecutando el método G");
    }
}
```

Realice las siguientes actividades:

a) Ejecute paso a paso el programa y observe cuidadosamente su funcionamiento.
b) ¿Qué salida produce por Consola?
c) ¿En qué orden se invocan los métodos contenidos en un delegado?

4) Modifique el método Main() de la clase Program de la siguiente manera:

```
public static void Main()
 {
     Trabajador o=new Trabajador();
     o.Trabajando = new TrabajandoEventHandler(F);
     o.Trabajando += new TrabajandoEventHandler(G);
     o.Trabajando += new TrabajandoEventHandler(G);
     o.Trabajando += new TrabajandoEventHandler(G);
     Console.WriteLine("Todo listo para comenzar");
     o.Trabajar();
     Console.ReadKey(true);
 }
```

Realice las siguientes actividades:

a) Ejecute paso a paso el programa y observe cuidadosamente su funcionamiento.
b) ¿Intentar que un delegado contenga más de una instancia del mismo método produce algún error o es una posibilidad perfectamente válida?
c)¿ En caso de funcionar, qué salida produce por Consola?
d) Borre la instrucción "`o.Trabajando = new TrabajandoEventHandler(F);`" del método Main() y ejecute el programa nuevamente. Observe que no se produce error alguno. ¿Por qué no? Piense cuál es el valor de `o.Trabajando` en la primera instrucción ¿Puede decir entonces que, cuando uno de los operandos es un delegado, el operador "+"está sobrecargado?

5) Compruebe si "`o.Trabajando = o.Trabajando + new TrabajandoEventHandler(G)`" es equivalente a la forma abreviada "`o.Trabajando += new TrabajandoEventHandler(G)`" Codifique el método Main() de la siguiente manera:

```
public static void Main()
 {
     Trabajador o=new Trabajador();
     o.Trabajando = new TrabajandoEventHandler(F);
     o.Trabajando += new TrabajandoEventHandler(G);
     o.Trabajando += new TrabajandoEventHandler(G);
     o.Trabajando = new TrabajandoEventHandler(G);
     Console.WriteLine("Todo listo para comenzar");
     o.Trabajar();
     Console.ReadKey(true);
 }
```

¿Cuál es la salida por la Consola?

6) Codifique el método Main() de la siguiente manera

```
public static void Main()
 {
     Trabajador o=new Trabajador();
     o.Trabajando = new TrabajandoEventHandler(F);
     o.Trabajando += new TrabajandoEventHandler(G);
     o.Trabajar();
     Console.WriteLine("---------------");
     o.Trabajando -= new TrabajandoEventHandler(F);
     o.Trabajar();
     Console.ReadKey(true);
 }
```

a) ¿Cuál es la salida por Consola?
b) Qué hace el operador "`-=`"
c) De qué otra forma puede escribir la línea

```
o.Trabajando -= new TrabajandoEventHandler(F);
```

d) En el caso que el delegado `o.Trabajando` posea varias instancias del método `F()` ¿Cuántas son eliminadas con la siguiente instrucción?

```
o.Trabajando -= new TrabajandoEventHandler(F);
```

7) Codifique el método Main() de la siguiente manera:

```
public static void Main()
 {
     Trabajador o=new Trabajador();
     o.Trabajando -= new TrabajandoEventHandler(F);
     o.Trabajar();
     Console.ReadKey(true);
 }
```

a) ¿Cuál es la salida por Consola?
b) ¿Qué puede decir del operador "-" (observe su respuesta 4d respecto del operador "+")

8) Hasta ahora utilizamos el operador "+" para agregar un único método a la lista de métodos de un delegado. Cuál es el resultado de la aplicación del operador "+" entre dos delegados que poseen una lista con varios métodos. Para ello, codifique, ejecute y analice el siguiente programa:

```
using System;
delegate void TrabajandoEventHandler();
class Program
{
    private static TrabajandoEventHandler delegado1,delegado2;
    public static void Main()
    {
        delegado1 += new TrabajandoEventHandler(F);
        delegado1 += new TrabajandoEventHandler(G);
        delegado2 += new TrabajandoEventHandler(F);
        delegado2 += new TrabajandoEventHandler(G);
        Trabajador o=new Trabajador();
        o.Trabajando = delegado1+delegado2;
        o.Trabajar();
        Console.ReadKey(true);
    }
    private static void F(){
        Console.WriteLine("El trabajo se ha iniciado");
    }
    private static void G(){
        Console.WriteLine("Ejecutando el método G");
    }
}
class Trabajador{
    public TrabajandoEventHandler Trabajando;
    public void Trabajar(){
        if (Trabajando != null) Trabajando();
        //realiza algún trabajo
        Console.WriteLine("Trabajo concluido");
    }
}
```

9) Hasta ahora utilizamos el operador "-" para eliminar un único método de la lista de métodos de un delegado. Cuál es el resultado de la aplicación del operador "-" entre dos delegados que poseen una lista con varios métodos cada uno. Para ello, codifique, ejecute y analice un programa adecuado contestando las siguientes preguntas:

a) Si `delegado1` posee los métodos A, B y C (en ese orden) y `delegado2` posee los métodos B, C y D (en ese orden), qué métodos serán asignados a `delegado3` al ejecutar la siguiente instrucción `delegado3=delegado1-delegado2;`

b) Si `delegado1` posee los métodos A, B, C y D (en ese orden) y `delegado2` posee los métodos A y C (en ese orden), qué métodos serán asignados a `delegado3` al ejecutar la siguiente instrucción `delegado3=delegado1-delegado2;`

c) Si `delegado1` posee los métodos A, B, C y D (en ese orden) y `delegado2` posee los métodos A, B y C (en ese orden), qué métodos serán asignados a `delegado3` al ejecutar la siguiente instrucción `delegado3=delegado1-delegado2;`

d) Si `delegado1` posee los métodos A, B, C y D (en ese orden) y `delegado2` posee los métodos B y C (en ese orden), qué métodos serán asignados a `delegado3` al ejecutar la siguiente instrucción `delegado3=delegado1-delegado2;`

10) Cree una nueva solución en el SharpDevelop y codifique el siguiente programa:

```
using System;
delegate void tipoDelegado();

class Program{
	public static void Main(){
		tipoDelegado delegado;
		Objeto o=new Objeto();
		delegado = o.getDelegado("999");
		delegado();
		delegado = o.getDelegado("123");
		delegado();
		Console.ReadKey();
	}
}

class Objeto{
	private void metodoPrivado(){
		Console.WriteLine("método privado del objeto");
	}
	public void metodoPublico(){
		Console.WriteLine("método público del objeto");
	}
	public tipoDelegado getDelegado(string clave){
		if (clave=="123") return new tipoDelegado(metodoPrivado);
		else return new tipoDelegado(metodoPublico);
	}
}
```

a) Cuál es la salida por consola?

Obsérvese que, cuando se realiza una llamada a través de un objeto delegado no se tienen en cuenta los modificadores de visibilidad de los métodos que se ejecutarán, lo que permite llamar desde un tipo a métodos privados de otros tipos que estén almacenados en un delegado.

11) Cree una nueva solución en el SharpDevelop y codifique el siguiente programa:

```
using System;
delegate int tipoDelegado();

class Program{
	public static void Main(){
		tipoDelegado delegado;
		delegado=new tipoDelegado(devuelveUno);
		delegado+=new tipoDelegado(devuelveDos);
		int i= delegado();
		Console.WriteLine(i);
		Console.ReadKey();
	}

	static int devuelveUno(){
		Console.WriteLine("Ejecutando devuelveUno()");
		return 1;
	}

	static int devuelveDos(){
		Console.WriteLine("Ejecutando devuelveDos()");
		return 2;
	}
}
```

a) Cuál es la salida que produce por Consola?

b) Qué puede decir sobre los valores al invocar los métodos “encolados” en un delegado?

Nota: Observe que en la instrucción “`delegado=new tipoDelegado(devuelveUno);`” no se ha utilizado el operador “+=” como veníamos haciendo en ejemplos anteriores. Esto se debe a que delegado se ha definido como una variable local en el método Main(), por lo tanto el compilador no permite expresiones de lectura sobre esta variable verificando que nunca fue asignada.

12) Reemplaza el método Main() por el siguiente:

```
public static void Main(){
      tipoDelegado delegado;
      delegado=new tipoDelegado(devuelveUno);
      delegado+=new tipoDelegado(devuelveDos);
      System.Delegate[] listaDelegados=delegado.GetInvocationList();
      foreach(tipoDelegado del in listaDelegados){
            int i=del();
            Console.WriteLine(i);
      }
      Console.ReadKey();
}
```

a) Cuál es la salida que produce por Consola?
b) Qué hace el método `.GetInvocationList()`
c) Serviría este método para poder ejecutar los métodos encolados en un delegado en un orden distinto al de la cola?

13) Cree una nueva solución en el SharpDevelop y codifique el siguiente programa:

```
using System;
delegate void TicEventHandler(DateTime hora);
class program{
      static int cont=0;
      static Clock reloj=new Clock();
      static void Main(){
            reloj.Tic=new TicEventHandler(Tic);
            reloj.run();
      }
      private static void Tic(DateTime horaActual){
            Console.WriteLine(horaActual);
            cont++;
            if(cont==10) reloj.Detener();
      }
}
class Clock{
      private bool detener;
      public TicEventHandler Tic;
      public void run(){
            DateTime hora=DateTime.Parse("1/1/2000"),horaAux=DateTime.Now;
            detener=false;
            while (!detener){
                  if(hora.Second != horaAux.Second){
                        hora=horaAux;
                        if(Tic!=null) Tic(hora);
                  }
                  horaAux=DateTime.Now;
            }
      }
      public void Detener(){
            detener=true;
      }
}
```

a) ¿Qué hace el programa?
b) ¿Cuál es la salida por Consola?

14) Modifique el código del ejercicio 13 para cumplir con la convención vista en la teoría respecto de los parámetros que debe llevar un manejador de eventos (`sender` de tipo `object` y `e` de tipo `EventArgs` o clase derivada)

15) Analice el siguiente código:

```
using System;

class Program{
	public static void Main(){
		Palabras pal=new Palabras();
		pal.contar();
		Console.ReadKey();
	}
}

class Palabras{
	private int cantPalabras=0;
	public void contar(){
		Lector miLector=new Lector();
		miLector.padre = this;
		miLector.leer();
		Console.WriteLine("Cantidad de palabras leídas: {0}",cantPalabras);
	}
	public void unaMas(){
	 cantPalabras ++;
	}
}

class Lector{
	public Palabras padre;
	public void leer(){
		Console.WriteLine("Ingrese una palabra por línea");
		string st=Console.ReadLine();
		while (st!=""){
			padre.unaMas();
			st=Console.ReadLine();
		}
	}
}
```

Elimine la referencia circular (Palabra-Lector) utilizando delegados. Observe que el método `void unaMas()` puede convertirse en privado.

Nota: Se dice que dos objetos presentan una referencia circular cuando cada uno de ellos posee una referencia hacia el otro.