Microsoft .NET

PAV 2023

La Programación Básica en Entorno Visual (II)

UNIDAD N° 3

(Parte 2/3)

2023

CHAPTER 20

Decoupling application logic and handling events

https://docs.microsoft.com/es-es/dotnet/framework/winforms/index

Understanding delegates

Note Delegates are so named because they "delegate" processing to the referenced method when they are invoked.

```
Processor p = new Processor();
p.performCalculation();
```

```
Processor p = new Processor();
delegate ... performCalculationDelegate ...;
performCalculationDelegate = p.performCalculation;
```

- A delegate is equivalent to a type-safe function pointer or a callback
- Delegates can be bound to a single method or to multiple methods, referred to as multicasting
- A multicast delegate maintains an invocation list of the methods it is bound to
- The binding mechanism used with delegates is dynamic: a delegate can be bound at run time to any method whose signature matches
- Delegates have many uses in the .NET Framework (events and lambda expressions)

Código

Enabling notifications by using events

An event is an action which you can respond to, or "handle," in code. Events can be generated by a user action, such as clicking the mouse or pressing a key; by program code; or by the system.

Event-driven applications execute code in response to an event. Each form and control exposes a predefined set of events that you can program against. If one of these events occurs and there is code in the associated event handler, that code is invoked.

This event model uses *delegates* to bind events to the methods that are used to handle them. The delegate enables other classes to register for event notification by specifying a handler method. When the event occurs, the delegate calls the bound method

Declaring an event

```
event delegateTypeName eventName
```

```
class TemperatureMonitor
{
    public delegate void StopMachineryDelegate();
    ...
}
```

You can define the *MachineOverheating* event, which will invoke the *stopMachineryDelegate*, like this:

```
class TemperatureMonitor
{
     public delegate void StopMachineryDelegate();
     public event StopMachineryDelegate MachineOverheating;
     ...
}
```

Subscribing to an event

```
tempMonitor.MachineOverheating += (() => { folder.StopFolding(0); });
tempMonitor.MachineOverheating += welder.FinishWelding;
tempMonitor.MachineOverheating += painter.PaintOff;
```

Unsubscribing from an event

```
tempMonitor.MachineOverheating -= (() => { folder.StopFolding(0); });
tempMonitor.MachineOverheating -= welder.FinishWelding;
tempMonitor.MachineOverheating -= painter.PaintOff;
```

Raising an event

```
class TemperatureMonitor
{
    public delegate void StopMachineryDelegate();
    public event StopMachineryDelegate MachineOverheating;
    ...
    private void Notify()
    {
        if (this.MachineOverheating != null)
        {
            this.MachineOverheating();
        }
    }
    ...
}
```