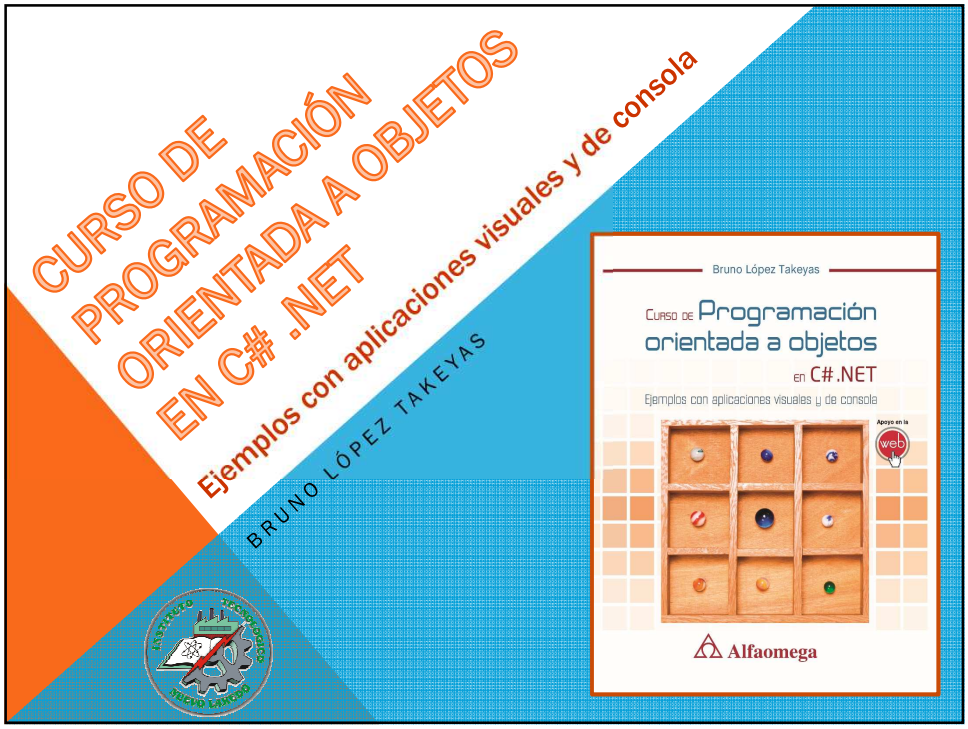

CURSO DE PROGRAMACIÓN ORIENTADA A OBJETOS EN C# .NET
Ejemplos con aplicaciones visuales y de consola
BRUNO LÓPEZ TAKEYAS
Bruno López Takeyas
Curso de Programación orientada a objetos en C#.NET
Ejemplos con aplicaciones visuales y de consola
Alfaomega


CAPÍTULO 8
EVENTOS
2

Preguntas detonadoras

- ¿Qué es un evento? ¿Para qué sirve?
- ¿Qué temas se deben dominar para implementar aplicaciones con eventos?
- ¿Qué se requiere para que una clase genere eventos?
- ¿Qué se requiere para que un objeto reciba notificaciones de eventos?

3

Evento

- Mecanismo mediante el cual una clase puede proporcionar notificaciones a sus clientes cuando ocurre algún suceso importante con sus objetos.
- Es generado por una clase publicadora y notificado a un conjunto de objetos de clases clientes o suscriptoras.

4

¿Dónde usar eventos?

- La implementación más común de eventos se presenta en aplicaciones visuales.
- Las clases de los controles de la interfaz gráfica disponen de eventos que se notifican cuando el usuario realiza alguna actividad con el control (como hacer *click* en algún botón de la forma).
- Los eventos no son exclusivos para las aplicaciones visuales, ya que una aplicación de consola puede tener clases que disponen de eventos, además que el programador puede diseñar sus propios eventos.

5

Uso de eventos

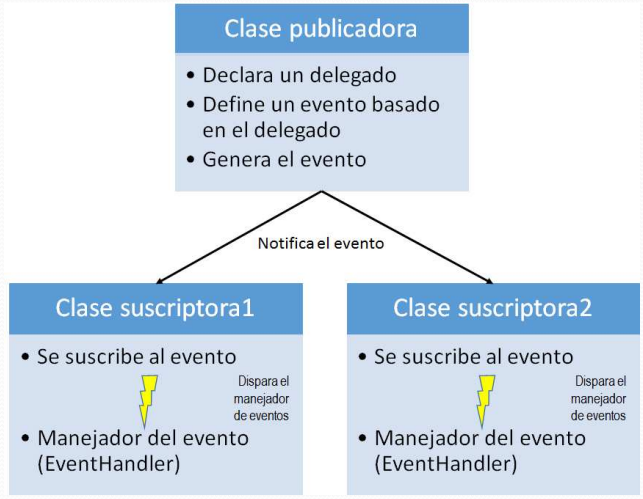
Requisitos:

1. *Clase cuyos objetos generan el evento (**publicadora**).*
2. *Las clases cuyos objetos reciben el evento (**suscriptoras**).*

- Los eventos se declaran a través de delegados
- Cuando se produce el evento, entonces se llama a los delegados que proporcionan las clases suscriptoras para dicho evento

6

Diseño de eventos



7

Declaración de un evento

La clase publicadora debe:

1. *Definir un delegado para dicho evento.*
   - *EventHandler*

```
// Delegado definido por el programador
public delegate void DelegadoEventHandler(string strMensaje);
```

2. *Utiliza el delegado para definir el evento*

```
// Evento
public event DelegadoEventHandler Evento;
```

8

Generación de un evento

La clase publicadora genera el evento:

```
// Delegado definido por el programador
public delegate void DelegadoEventHandler(string strMensaje);

// Definición de la clase que genera el evento (clase publicadora)
class ClasePublicadora
{
    // Atributos, métodos y propiedades
       . . . . .

    // Evento
    public event DelegadoEventHandler Evento;

    // Propiedad que genera el evento
    public int Propiedad
    {
       set
       {
          // Modifica el valor de su atributo
             . . . . .
          // Genera el evento y dispara una notificación
          this.Evento("Mensaje generado por el evento");
       }
    }
}
```

9

Suscripción a un evento (opción 1)

La clase suscriptora debe:

1. *Utilizar el operador += para recibir notificaciones*
2. *Proporciona un delegado con el método gestor al evento*

```
// Suscripción al evento

miObjeto.Evento+= new DelegadoEventHandler(MetodoGestor);
```

10

Suscripción a un evento (opción 2)

La clase suscriptora debe:

1. *Utilizar el operador += para recibir notificaciones*
2. *Proporciona el método gestor al evento (sin usar el delegado).*

```
// Otra forma de suscripción al evento

miObjeto.Evento += MetodoGestor;
```

11

Suscripción a un evento (opción 3)

La clase suscriptora debe:

1. *Utilizar el operador += para recibir notificaciones*
2. *Proporciona una expresión lambda al evento (sin usar el delegado ni el método gestor).*

```
// Otra forma de suscripción al evento mediante una expresión lambda

miObjeto.Evento += (Mensaje) => { Console.WriteLine(Mensaje); };
```

12

Ejemplo de aplicación de eventos

- Un banco desea enviar una notificación a sus clientes cuando se modifique el saldo de su cuenta.

- Datos de la cuenta bancaria
  - *Cuenta (string)*
  - *Cliente (string)*
  - *Saldo (double)*

- Métodos
  - *Constructor()*
  - *Depositar()*
  - *Retirar()*

13

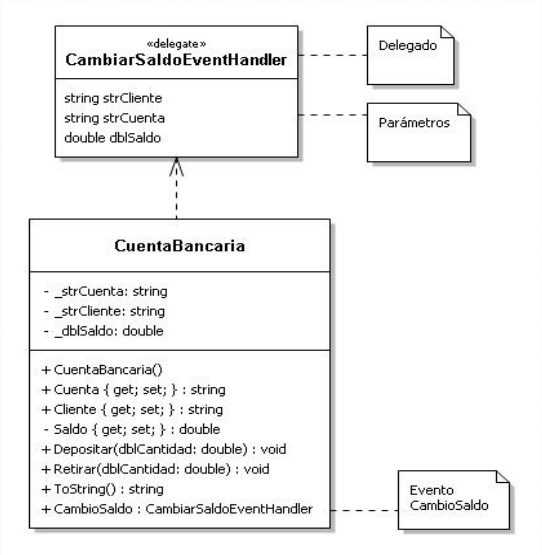
Operaciones en la cuenta bancaria

- Cuando se realiza una operación (depósito o retiro) de una cuenta bancaria, esta clase genera un evento.

- Para ello, la clase define un delegado y su evento:
  - ***CambiarSaldoEventHandler****:* Delegado con tres parámetros (cuenta, cliente y saldo).
  - ***CambioSaldo****:* Evento que se genera cuando se modifica el saldo de la cuenta.

14

Diseño de la clase



15

Definiciones del delegado y evento

```
// Delegado
public delegate void CambiarSaldoEventHandler(string
strCliente, string strCuenta, double dblSaldo);

// Evento
public event CambiarSaldoEventHandler CambioSaldo;
```

16

Generación del evento al depositar

```
// Método público para depositar dinero en la cuenta
public void Depositar(double dblCantidad)
{
    // Valida la cantidad
    if (dblCantidad > 0)
    {
       // Incrementa el saldo
       this.Saldo = this.Saldo + dblCantidad;

       // Genera el evento
       CambioSaldo(this.Cliente, this.Cuenta, this.Saldo);
    }
    else
        throw new Exception("Cantidad inválida !!!");
}
```

17

Generación del evento al retirar

```
// Método público para retirar dinero de la cuenta
public void Retirar(double dblCantidad)
{
   // Valida si hay saldo suficiente ...
   if (this.Saldo >= dblCantidad)
   {
       // Reduce el saldo
       Saldo = Saldo - dblCantidad;

       // Genera el evento
       CambioSaldo(this.Cliente,this.Cuenta,this.Saldo);
   }
   else
       throw new Exception("Saldo insuficiente !!!");
}
```

18

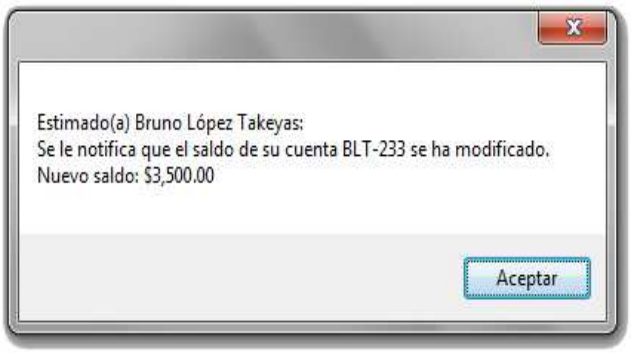
Implementación del método gestor y suscripción al evento

```
// Método gestor ejecutado al modificar el saldo
public void MetodoGestor(string strCliente, string strCuenta,
double dblSaldo)
{
   MessageBox.Show("Estimado(a) " + strCliente + ":\nSe le
notifica que el saldo de su cuenta " + strCuenta + " se ha
modificado.\nNuevo saldo: " + dblSaldo.ToString("C"));
}
```

```
// Suscripción al evento
miCuentaBancaria.CambioSaldo += MetodoGestor;
```

19

Ejemplo de ejecución



20

Cancelar la suscripción a un evento

- Se puede impedir que se invoque el manejador de eventos, cuando se genera una notificación.
- Debe cancelar la suscripción antes de eliminar el objeto suscriptor.
- Se utiliza el operador -= para cancelar la suscripción.
- *Ejemplo:*
  - `miObjeto.Evento -= MetodoGestor;`

21

La interfase INotifyPropertyChanged

- Incluida en el espacio de nombres:
  - `using System.ComponentModel;`
- Se utiliza para notificar a clases suscriptoras sobre el cambio de valor de alguna propiedad de la clase publicadora.
- Su uso más común es mediante enlace de clientes (binding)

22

Sintaxis de INotifyPropertyChanged

- `public interface INotifyPropertyChanged`

23

Implementación de INotifyPropertyChanged

- La clase publicadora debe:

1. *Definir un evento de nombre* **`PropertyChanged`** *de tipo* **`PropertyChangedEventHandler`** *de la interfase* **`INotifyPropertyChanged`**
2. *Implementar un método que dispare el evento.*
3. *Invocar el método disparador del evento al modificar el valor de una propiedad*

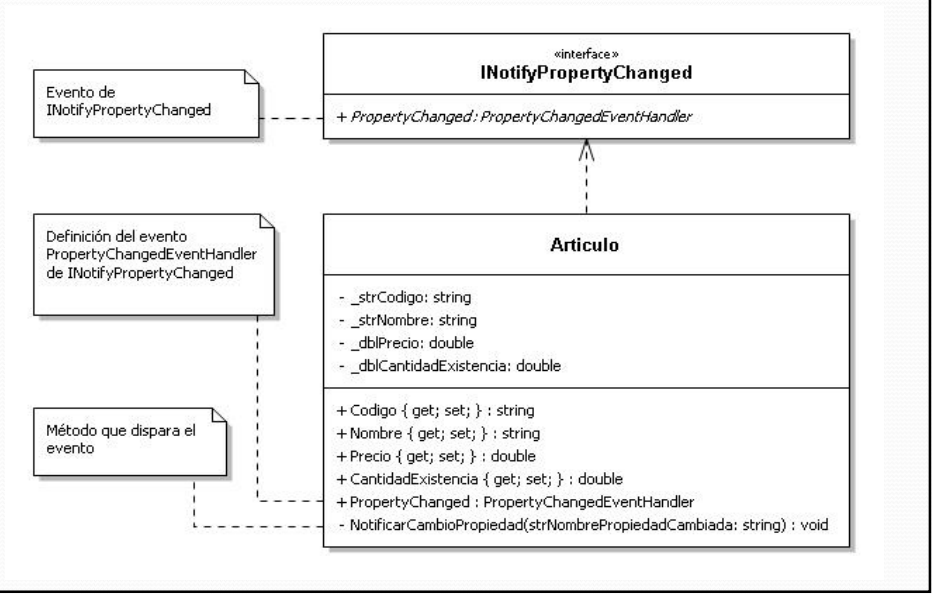
24

Ejemplo de INotifyPropertyChanged

- Una clase llamada Artículo tiene 4 atributos con sus respectivas propiedades
  - *Código (string)*
  - *Nombre (string)*
  - *Precio (double)*
  - *Cantidad en existencia (double)*
- Requiere enviar una notificación a sus suscriptores cuando se modifique el valor de alguna de sus propiedades

25

Diseño de la clase



Definición del evento y método disparador en la clase publicadora

```
class Articulo : INotifyPropertyChanged
{
  // Definiciones de atributos y propiedades
  . . .
  . . .

// Definición del evento PropertyChanged de INotifyPropertyChanged
  public event PropertyChangedEventHandler PropertyChanged;

// Método disparador del evento
  private void NotificarCambioPropiedad(string
strNombrePropiedadCambiada)
  {
     if (PropertyChanged != null)
       PropertyChanged(this, new
PropertyChangedEventArgs(strNombrePropiedadCambiada));
  }
}
```

¿Cómo disparar el evento en la clase publicadora?

- Cada propiedad de la clase publicadora invoca el método disparador (enviándole su nombre)

```
// Propiedad
public string  Codigo
{
   get { return _strCodigo; }
   set
   {
      if (value == "")
         throw new Exception("No deje en blanco el código del artículo");
      else
      {
         _strCodigo = value;
         NotificarCambioPropiedad("Codigo");
      }
   }
}
```

28

Método gestor

- Se invoca al suscribirse al evento de la clase publicadora
- *Se codifica fuera de la clase suscriptora y/o de la clase publicadora*
- *P. ejem. En la forma*

```
private void MetodoGestor(object sender, PropertyChangedEventArgs e)
{
   MessageBox.Show("Se ha cambiado el valor de "+e.PropertyName);
}
```

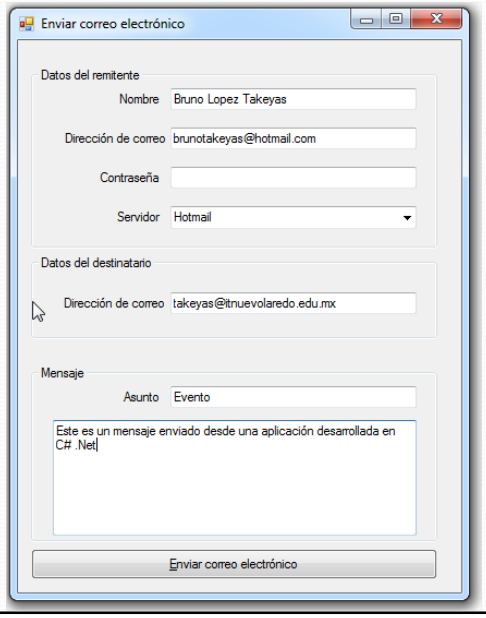
29

Suscribirse al evento

- *Se suscribe al manejador del evento PropertyChangedEventHandler de la clase publicadora y se envía como parámetro el nombre del método gestor*

```
Articulo miArticulo = new Articulo();

// El objeto miArticulo se suscribe al evento
miArticulo.PropertyChanged += new
PropertyChangedEventHandler(MetodoGestor);

miArticulo.Codigo = “JJGH-8998";
miArticulo.Nombre = "Jugo de arándanos";
miArticulo.Precio = 123.55;
miArticulo.CantidadExistencia = 2345;
```

¿Cómo enviar un email desde una aplicación en C#?



31

Espacios de nombres requeridos

```
// Para poder enviar el correo
using System.Net.Mail;

// Para definir las credenciales del remitente del email
using System.Net;

// Para enviar archivos adjuntos
using System.Net.Mime;
```

32

La clase MailMessage

- Requiere el espacio de nombres
  - `using System.Net.Mail;`
- Se utiliza para enviar un correo electrónico por medio de la clase `SmtpClient`

33

Principales métodos y propiedades de la clase MailMessage

Método o propiedad	Uso
`Subject`	Asunto del mensaje del correo electrónico
`To`	Colección de direcciones de destinatarios
`From`	Dirección del remitente
`Body`	Mensaje (cuerpo) del texto del correo electrónico
`Attachments`	Colección de archivos adjuntos

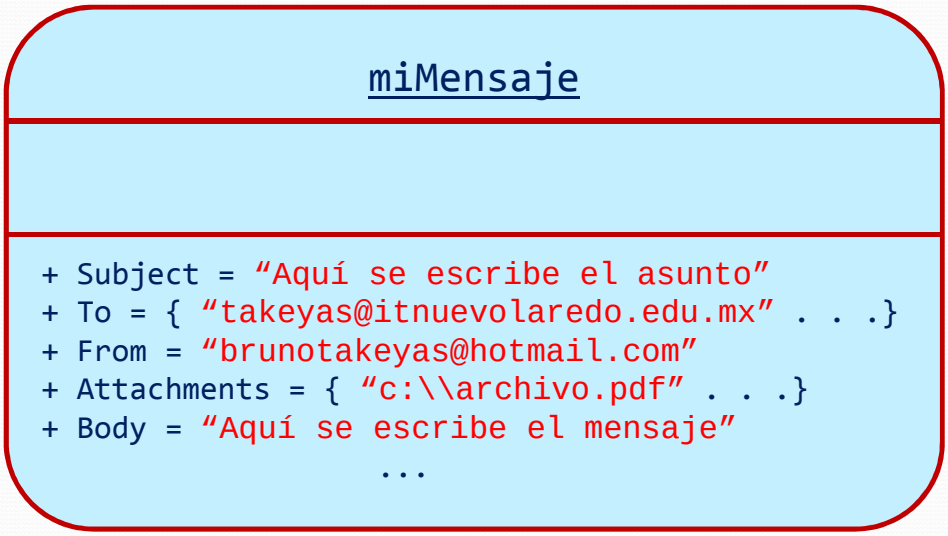
34

Ejemplo de uso de MailMessage

```
using System.Net.Mail;

MailMessage miMensaje = new MailMessage();

miMensaje.Subject = “Aquí se escribe el asunto”;

miMensaje.To.Add(new
MailAddress(“takeyas@itnuevolaredo.edu.mx”));

miMensaje.From = new
MailAddress(“brunotakeyas@hotmail.com”,”Bruno Lopez
Takeyas”);

// Si desea adjuntar algún archivo . . .
miMensaje.Attachments.Add(new Attachment("C:\\archivo.pdf"));

miMensaje.Body = “Aquí se escribe el mensaje";
```

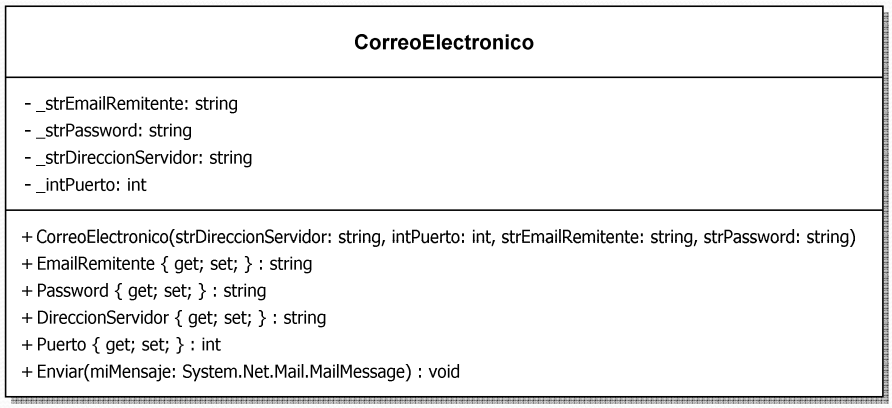
35

El objeto miMensaje



36

La clase CorreoElectronico



- Las propiedades `Password` y `Puerto` presentan un nivel asimétrico de acceso (`public set, private get`) para mayor seguridad de la constraseña y el puerto

37

El método para Enviar el mensaje

```
public void Enviar(System.Net.Mail.MailMessage miMensaje)
{
  System.Net.Mail.SmtpClient miCliente = new
System.Net.Mail.SmtpClient(DireccionServidor, Puerto);

  // Autenticación con el servidor
  miCliente.Credentials = new
System.Net.NetworkCredential(EmailRemitente, Password);
 // Establece una conexión segura
  miCliente.EnableSsl = true;
 // Envía el correo electrónico
  miCliente.Send(miMensaje);
}
```

- El método `Enviar()` se implementa dentro de la clase `CorreoElectronico`

38

```
class CorreoElectronico
    {
        public CorreoElectronico(string strDireccionServidor, int intPuerto, string strEmailRemitente, string strPassword) {
            DireccionServidor = strDireccionServidor;
            Puerto = intPuerto;
            EmailRemitente = strEmailRemitente;
            Password = strPassword;
        }

        private string _strEmailRemitente;
        public string EmailRemitente   {
            get { return _strEmailRemitente; }
            set             {
                if (value == "")
                    throw new Exception("No deje en blanco la dirección de correo electrónico del remitente");
                else
                    _strEmailRemitente = value;
            }
        }

        private string _strPassword;
        public string Password        {
            set             {
                if (value == "")
                    throw new Exception("No deje en blanco la contraseña del correo electrónico del remitente");
                else
                    _strPassword = value;
            }
            private get { return _strPassword; }
        }

        private string _strDireccionServidor;
        private string DireccionServidor        {
            get { return _strDireccionServidor; }
            set             {
                if (value == "")
                    throw new Exception("No deje en blanco la dirección del servidor SMTP");
                else
                    _strDireccionServidor = value;
            }
        }

        private int _intPuerto;
        public  int Puerto        {
            set             {
                if (value <= 0)
                    _intPuerto = 25;
                else
                    _intPuerto = value;
            }
            private get { return _intPuerto; }
        }

        public void Enviar(System.Net.Mail.MailMessage miMensaje)        {
            System.Net.Mail.SmtpClient miCliente = new System.Net.Mail.SmtpClient(DireccionServidor, Puerto);

            // Autenticacion con el servidor
            miCliente.Credentials = new System.Net.NetworkCredential(EmailRemitente, Password);
            // Establece una conexión segura
            miCliente.EnableSsl = true;
            // Envia el mensaje
            miCliente.Send(miMensaje);
        }
    }
```

Detectar el servidor remitente

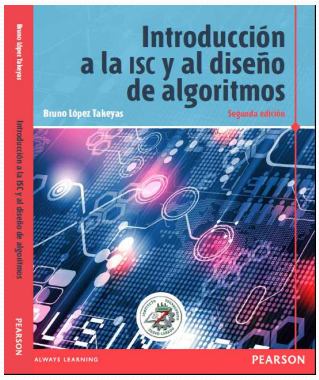
```
private void DetectarDireccionServidor(out string
strDireccionServidor, out int intPuerto)
{
/* Cliente SMTP
 * Gmail   : smtp.gmail.com  puerto:587
 * Hotmail : smtp.live.com   puerto:25
 * Yahoo ! : smtp.yahoo.com                          */

 switch (cboNombreServidor.Text)
 {
   case "Hotmail": intPuerto = 25;
          strDireccionServidor = "smtp.live.com"; break;
   case "Yahoo !": intPuerto = 587;
          strDireccionServidor = "smtp.yahoo.com"; break;
   case "Gmail": intPuerto = 25;
          strDireccionServidor = "smtp.gmail.com"; break;

   default: throw new Exception("Servidor desconocido");
 }
}
```

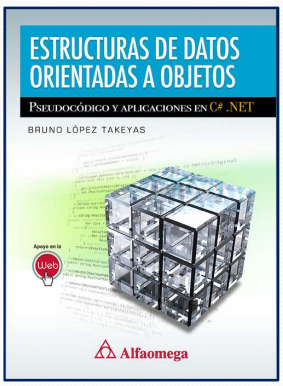
Método gestor que envía un email

```
public void MetodoGestor(string strCliente, string strCuenta, double
dblSaldo, string strEmailDestinatario)
{
 try {
  MailMessage miMensaje = new MailMessage();
  miMensaje.Subject = "Cambio de saldo";
  miMensaje.To.Add(new MailAddress(strEmailDestinatario));
  miMensaje.From = new MailAddress("brunotakeyas@hotmail.com");
  miMensaje.Body = "Estimado(a) " + strCliente + ":\nSe le notifica
que el saldo de su cuenta " + strCuenta + " se ha modificado.\nNuevo
saldo: " + dblSaldo.ToString("C");

  CorreoElectronico miCorreoElectronico = new
CorreoElectronico("smtp.live.com", 25, "brunotakeyas@hotmail.com",
txtPassword.Text);
  miCorreoElectronico.Enviar(miMensaje);
 }
 catch(Exception ex) {
    MessageBox.Show(ex.Message);
 }
}
```

Otros títulos del autor

http://www.itnuevolaredo.edu.mx/Takeyas/Libro





takeyas@itnuevolaredo.edu.mx

Bruno López Takeyas