UNIDAD N° 3

EJERCICIO N° 2

AGREGANDO EVENTOS

Para realizar este ejercicio es necesario realizar el Ejercicio N° 1 de esta unidad.

1. Abrir la Solución **Delegates** descargada en el Ejercicio N° 1.
2. En el proyecto AuditService, abrir el archivo Auditor.cs.
3. Agregar un delegado público llamado *AuditingCompleteDelegate* en la clase *Auditor*. Este delegado especifica un método que recibe un string y retorna *void*. El código, en negrita, lo muestra:

```
class Auditor
{
      public delegate void AuditingCompleteDelegate(string
      message);
      ...
}
```

4. Agregar un evento público llamado *AuditProcessingComplete* a la clase *Auditor*, después del delegado *AuditingCompleteDelegate*. Este evento estará basado en el mismo delegado, como se muestra:

```
class Auditor
{
      public delegate void AuditingCompleteDelegate(string
      message);
      public event AuditingCompleteDelegate
      AuditProcessingComplete;
      ...
}
```

5. Ubicar el método AuditOrder. Este es el método que se ejecuta utilizando el delegado en el objeto CheckoutController. A su vez invoca otro método privado llamado *DoAuditing* para realmente realizar la operación de auditoría. El método se ve así:

```
public void AuditOrder(Order order)
{
     DoAuditing(order);
}
```

6. Desplazar hacia abajo, hasta el método *DoAuditing*. El código en este método está encerrado en un bloque *try / catch*; utiliza las API XML de la biblioteca de clases de .NET Framework para generar una representación XML del pedido que se audita y la guarda en un archivo (los detalles exactos de cómo funciona va más allá del alcance de este ejercicio).

   Después del bloque *catch* agregar un bloque *finally* que invocará al evento *AuditProcessingComplete*, como se muestra en negrita:

```
private async void doAuditing(Order order)
{
     ...
     if (ageRestrictedItems.Count > 0)
     {
          try
          {
               ...
          }
          catch (Exception ex)
          {
               ...
          }
          finally
          {
               if (AuditProcessingComplete != null)
               {
               AuditProcessingComplete(string.Format(
               "Audit record written for Order{0}",
               order.OrderID));
               }
          }
     }
}
```

7. En el proyecto DeliveryService, abrir el archivo Shipper.cs.
8. Agregar el delegado público *ShippingCompleteDelegate* a la clase *Shipper*. Este delegado especifica un método que recibe un string y retorna *void*. Como se muestra:

```
class Shipper
{
     public delegate void ShippingCompleteDelegate(string
     message);
     ...
}
```

9. Agregar un evento público llamado *ShipProcessingComplete* a la clase *Auditor*, después del delegado *ShippingCompleteDelegate*. Este evento estará basado en el mismo delegado, como se muestra:

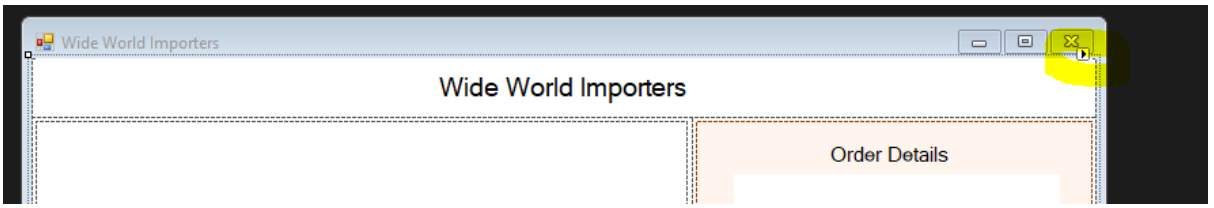
```
class Auditor
{
     public delegate void ShippingCompleteDelegate(string
     message);
     public event ShippingCompleteDelegate
     ShipProcessingComplete;
     ...
}
```

10. Ubicar el método *DoShipping*, el cual posee la lógica para el envío. En el método, después del bloque *catch*, agregar el bloque *finally* que invoca al evento *ShipProcessingComplete*, como se muestra en negrita:

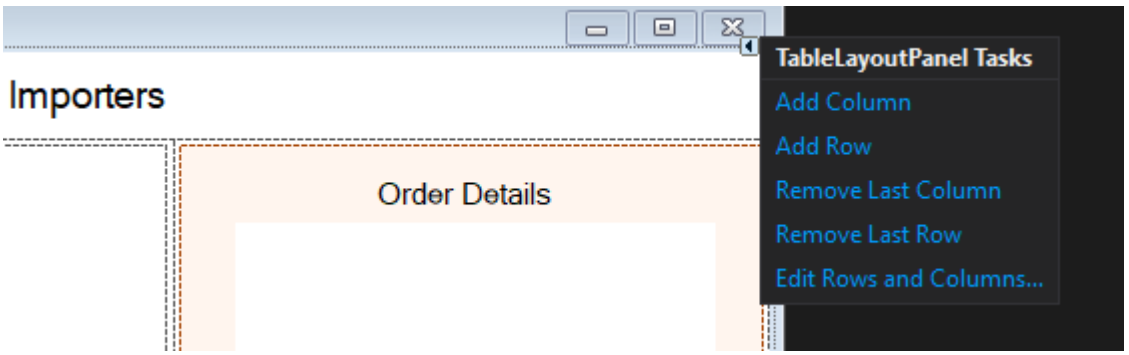
```
private async void DoShipping(Order order)
{
     try
     {
          ...
     }
     catch (Exception ex)
     {
          ...
     }
     finally
     {
          if (ShipProcessingComplete != null)
          {
               ShipProcessingComplete(string.Format(
```

```
                    "Dispatch note generated for Order {0}",
                    order.OrderID));
            }
        }
    }
```

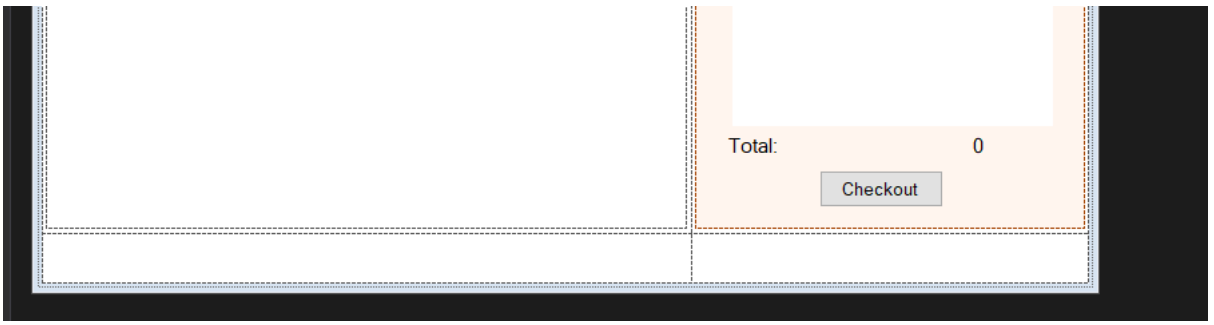
11. Ir al proyecto DelegatesForms y visualizar el formulario en el diseñador.
12. Visualizar la ventana Esquema del Documento (Document Outline). Se encuentra entre las ventanas flotantes a la izquierda del Visual Studio, en caso que no se encuentre ir al menú Ver (View), buscar Otras Ventanas (Other Windows) y elegir Esquema del Documento (Document Outline). También se puede visualizar utilizando la combinación de teclas *Ctrl + W, U.*
13. En el Esquema del Documento (Document Outline) buscar el control *tlpBase* del tipo TabletLayoutPanel. En el formulario se selecciona el control y aparece un icono con la punta de una flecha como se muestra:



14. Hacer click en el icono. Se muestran varias opciones, elegir Agregar Fila (Add Row).



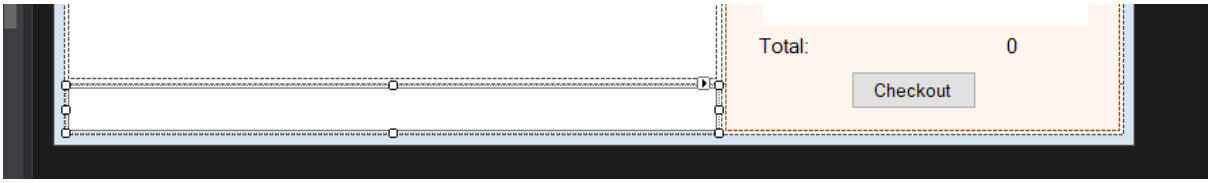
Al final del control se agrega una fila como se muestra:



Si se posiciona el cursor en la línea divisoria, haciendo click y desplazando el mouse se puede ajustar el alto de la fila.

15. En el Esquema del Documento ubicar el control pDetails. Presionar F4 para visualizar las propiedades. Ubicar la propiedad *RowSpan* y cambiar su valor a 2. El panel de la derecha en el formulario se ajusta como se muestra:

16. Agregar un TextBox en la nueva celda del control *tlpBase*. Nombrar al control *txtMessageBar*, cambiar la propiedad *Dock* a *Fill*, la propiedad *Multiline* a *true* y la propiedad *ScrollBars* a *Vertical*. Como se muestra a continuación:



17. Visualizar el código del formulario MainWindow.cs. Buscar el método *CheckoutButtonClicked* y quitar el código que muestra el resumen del pedido (Display a summary of the order). El bloque try/catch debe quedar como se muestra:

```
private void CheckoutButtonClicked(object sender, EventArgs e)
{
     try
     {
          // Perform the checkout processing
          ...

          //Clear out the order details so the user can start
          //again with a new order
          ...
     }
     catch (Exception ex)
     {
          ...
     }
}
```

18. Agregar un método privado llamado *DisplayMessage* a la clase *MainWindow*. Este método toma un parámetro string llamado *message*, y devuelve *void*. En el cuerpo de este método, agregar una instrucción para mostrar el valor de message en la propiedad *Text* del control *txtMessageBar*, seguido de un carácter de nueva línea, como se muestra a continuación en negrita:

```
private void DisplayMessage(string message)
{
      txtMessageBar.Text += message + "\n";
}
```

Esto provoca que se muestre el mensaje al pie del formulario.

19. Ubicar el constructor de la clase MainWindow y agregar el siguiente código en negrita:

```
public MainWindow()
{
      ...

      _auditor = new Auditor();
      _shipper = new Shipper();
      _checkoutController = new CheckoutController();

      ...

      _auditor.AuditProcessingComplete += DisplayMessage;
      _shipper.ShipProcessingComplete += DisplayMessage;
}
```

Estas declaraciones se suscriben a los eventos expuestos por los objetos *Auditor* y *Shipper*. Cuando los eventos se generan, se ejecuta el método *displayMessage*. Observar que el mismo método maneja ambos eventos.

20. Ejecutar la aplicación
21. Agregar productos al pedido y hacer click en Checkout. Visualizar el texto al pie del formulario.
22. Volver al Visual Studio y detener la ejecución.