UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL TUCUMÁN

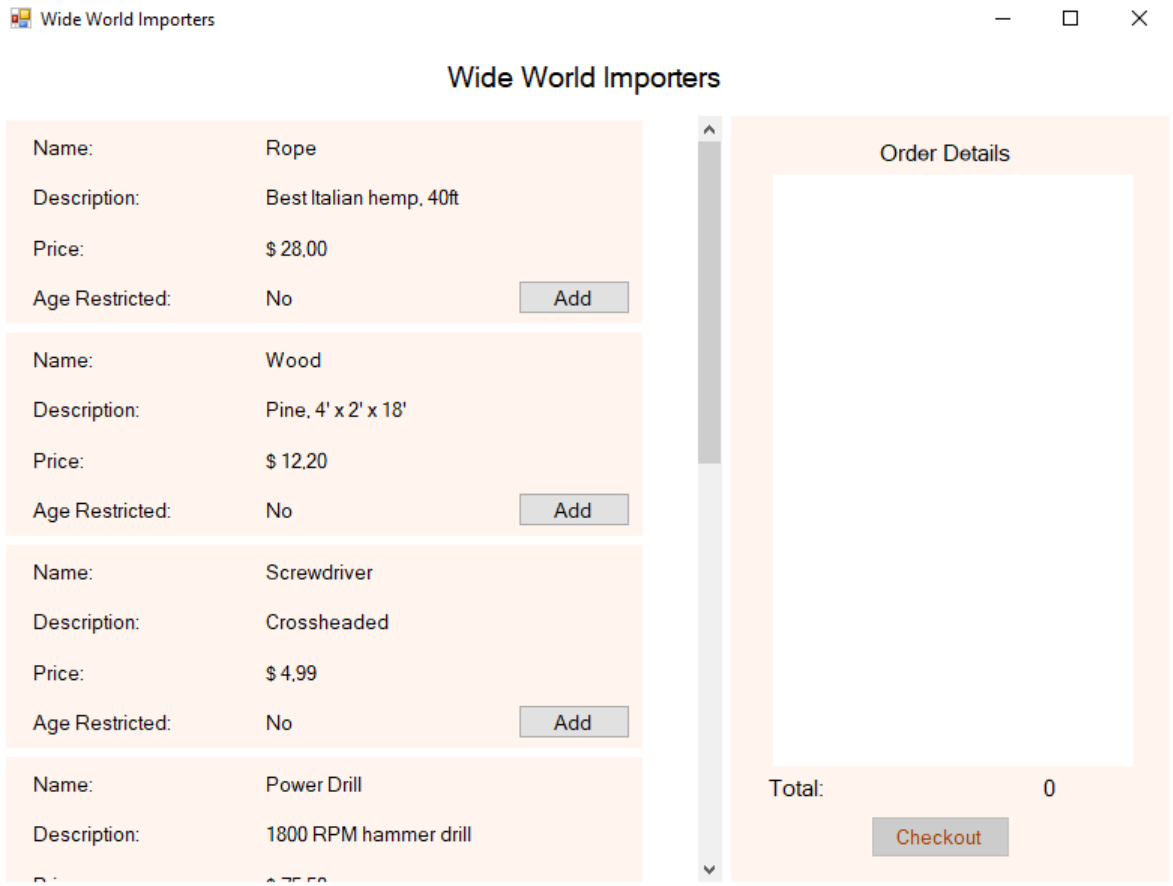
UNIDAD N° 3

EJERCICIO N° 1

TRABAJANDO CON DELEGADOS

Examinar la aplicación Wide World Importers

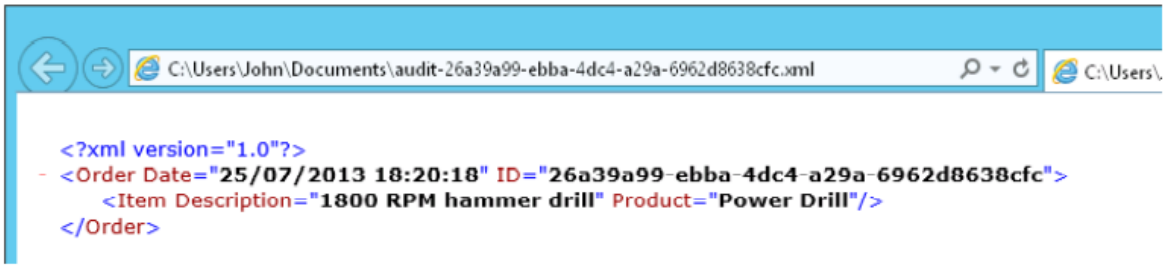
1. Descargar el ejemplo **Delegates** de:
   Acceso al archivo
2. Iniciar Visual Studio 2019
3. Ir al menú Archivo → Abrir → Proyecto / Solución
4. Ubicar la carpeta donde se descargó el ejemplo
5. Seleccionar la solución Delegates y elegir Abrir.
6. Ir al menú Depurar → Iniciar Depuración. Debería aparecer una ventana similar a:



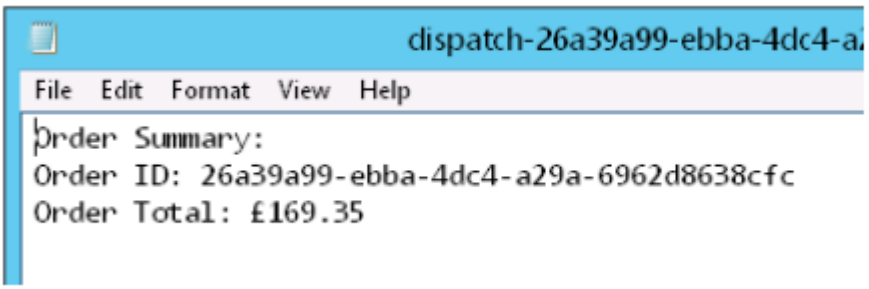
7. Seleccionar uno o más artículos y luego haga click en Add para incluirlos en el detalle del pedido. Estar seguro de seleccionar al menos un elemento con restricción de edad (age-restricted). A medida que se agrega un artículo, aparece en el panel de detalles (Order Details) en el lado derecho. Tener en cuenta que si se agrega el mismo artículo más de una vez, la cantidad se incrementa por cada click (esta versión de la aplicación no implementa la funcionalidad para eliminar elementos de la lista).
8. En el panel Detalles del pedido, hacer click en Checkout. Aparece un mensaje que indica que se ha realizado el pedido. El pedido recibe una identificación única y este ID se muestra junto con el valor del pedido.
9. Volver a Visual Studio y detener la ejecución.
10. Con el Explorador de archivos en Windows 8, 10 o el Explorador de Windows en Windows 7, ir a la carpeta de Documentos. Se debería ver dos archivos llamados *audit-nnnnnn.xml* (donde nnnnnn es el ID del pedido mostrado anteriormente) y *dispatch-nnnnnn.txt*. El primer archivo fue generado por el componente de auditoría, y el segundo archivo es la nota de envío generada por el componente de envío.

Note If there is no audit-*nnnnnn*.xml file, then you did not select any age-restricted items when you placed the order. In this case, switch back to the application and create a new order including one or more of these items.

11. Abrir el archivo *audit-nnnnnn.xml* utilizando el navegador. El contenido muestra la lista de productos con restricción de edad (age-restricted), con la fecha e identificador del pedido. Debería verse como:



12. Abrir el archivo *dispatch-nnnnnn.txt* con el bloc de notas. El contenido muestra el total del pedido, junto con el identificador. Debería verse como:



13. En Visual Studio, observar que la solución consta de los siguientes proyectos:
    - **Delegates**: Este proyecto contiene la aplicación en sí. El archivo MainWindow.cs define la interfaz de usuario y la lógica de la aplicación está contenida en el código de éste.
    - **AuditService**: Este proyecto contiene el componente que implementa el proceso de auditoría. Se empaqueta como una biblioteca de clases y contiene una sola clase llamada *Auditor*. Esta clase expone un único método público llamado *AuditOrder* que examina un pedido y genera el archivo *auditnnnnnn.xml* si el pedido contiene elementos restringidos por edad.
    - **DeliveryService**: Este proyecto contiene el componente que realiza la lógica de envío, empaquetado como una biblioteca de clases. La funcionalidad de envío está contenida en la clase *Shipper*, y proporciona un método público llamado *ShipOrder* que maneja el proceso de envío y también genera la nota de despacho.

Note You are welcome to examine the code in the *Auditor* and *Shipper* classes, but it is not necessary to fully understand the inner workings of these components in this application.

    - **DataTypes**: Este proyecto contiene los tipos de datos utilizados por los otros proyectos. La clase *Product* define los detalles de los productos que muestra la aplicación y los datos para los productos se mantienen en la clase *ProductDataSource* (la aplicación actualmente utiliza una colección de productos codificados. En un sistema de producción, esta información se recuperaría desde una base de datos o servicio web) Las clases *Order* y *OrderItem* implementan la estructura de un pedido; cada pedido contiene uno o más artículos de pedido.

14. En el proyecto DelegatesForms visualizar el código del formulario MainWindow.cs. Examinar los campos privados y el constructor. Lo importante es ver lo siguiente:

```
...
private Auditor _auditor = null;
private Shipper _shipper = null;

public MainWindow()
{
     ...
     _auditor = new Auditor();
     _shipper = new Shipper();
}
```

Los campos *_auditor* y *_shipper* contienen una referencia a las instancias de las clases *Auditor* y *Shipper*, el constructor inicializa estos objetos.

15. Ubicar el método *CheckButtonClicked*. Este método se ejecuta cada vez que el usuario presiona el botón Checkout. Debería verse como:

```
private void CheckoutButtonClicked(object sender, EventArgs e)
{
     try
     {
          // Perform the checkout processing
          if (requestPayment())
          {
               _auditor.AuditOrder(_order);
               _shipper.ShipOrder(_order);
          }
           ...
     }
      ...
}
```

Este método implementa el procesamiento del pago. Solicita el pago del cliente y luego invoca el método *AuditOrder* del objeto *_auditor* seguido por el método *ShipOrder* del objeto *_shipper*. Aquí se puede agregar cualquier lógica comercial adicional requerida en el futuro. El resto del código en este método después de la declaración *if* se refiere a la gestión de la interfaz de usuario: mostrar el cuadro de mensaje al usuario y borrar los detalles del pedido en el panel del lado derecho del formulario.

Aunque la aplicación funciona como se anuncia, los componentes Auditor y Shipper están estrechamente integrados con el proceso de pago. Si estos componentes cambian, la aplicación necesitará estar actualizada. Del mismo modo, si se necesita incorporar lógica adicional en el proceso de pago, posiblemente mediante el uso de componentes adicionales, se deberá modificar esta parte de la aplicación.

En los siguientes pasos se verá cómo es posible desacoplar el proceso de pago del negocio del resto de la aplicación.
El proceso de pago aún deberá invocar a los componentes Auditor y Shipper, pero debe ser lo suficientemente extensible para permitir que componentes adicionales se incorporen con facilidad. Esto se logrará creando un nuevo componente llamado *CheckoutController*, éste implementará la lógica de negocios para el proceso de pago y expondrá un delegado que permitirá que una aplicación especifique qué componentes y métodos deben incluirse

dentro de este proceso. El componente *CheckoutController* invocará estos métodos utilizando el delegado.

Crear el componente CheckoutController

1. En el Explorador de Soluciones hacer click con el botón derecho en la solución Delegates, desplazar hasta Agregar (Add) y elegir Nuevo Proyecto (New Project).

2. En el cuadro de diálogo Agregar Nuevo Proyecto (Add New Project), elegir Librería de Clases (.NET Framework) (Class Library). En el nombre escribir **CheckoutService**, y aceptar.

3. En el Explorador de Soluciones expandir el nuevo proyecto (CheckoutService) y a la clase agregada (Class1.cs) cambiarle el nombre por **CheckoutController.cs**. Visual Studio cambiará el nombre en todas las referencias a la clase.

4. Hacer click con el botón derecho en la carpeta References del proyecto CheckoutService y hacer click en Agregar Referencia (Add Reference).

5. En el cuadro de diálogo Administrador de Referencias, en el panel de la izquierda hacer click en Solución (Solution). En el panel del centro seleccionar el proyecto DataTypes y aceptar.
   La clase *CheckoutController* utilizará la clase *Order* definida en el proyecto DataTypes.

6. Agregar el siguiente código, en las directivas *using* de la clase *CheckoutController*:

```
using DataTypes;
```

7. Agregar el siguiente código, en negrita, a la clase *CheckoutController*:

```
public class CheckoutController
{
     public delegate void CheckoutDelegate(Order order);
}
```

   Se utilizará este tipo de delegado para hacer referencia a métodos que toman un parámetro *Order* y que no devuelvan un resultado. Esto simplemente coincide con la definición de los métodos *AuditOrder* y *ShipOrder* de las clases *Auditor* y *Shipper*.

8. Agregar un delegado público *CheckoutProcessing* basado en el tipo antes definido:

```
public class CheckoutController
{
      public delegate void CheckoutDelegate(Order order);
      public CheckoutDelegate CheckoutProcessing = null;
}
```

9. Visualizar el código del formulario MainWindow.cs, en el proyecto DelegatesForms y ubicar el método *RequestPayment*, cortar el método completo de la clase y pegarlo dentro de la clase *CheckoutController* del archivo CheckoutController.cs, debería verse así (en negrita lo nuevo):

```
public class CheckoutController
{
  public delegate void CheckoutDelegate(Order order);
  public CheckoutDelegate CheckoutProcessing = null;

  private bool requestPayment()
  {
   //Payment processing goes here
   //Payment logic is not implemented in this example
   //-simply return true to indicate payment has been received
   return true;
  }
}
```

10. Agregar el método *StartCheckoutProcess* a la clase *CheckoutController*:

```
.
public void StartCheckoutProcessing(Order order)
{
      // Perform the checkout processing
      if (RequestPayment())
      {
            if (CheckoutProcessing != null)
            {
                  CheckoutProcessing(order);
            }
      }
}
```

Este método proporciona la funcionalidad de pago implementada previamente por el método *CheckoutButtonClicked* de la clase MainWindow.

Solicita el pago y luego examina el delegado *CheckoutProcessing*; si este delegado no es nulo (se refiere a uno o más métodos), invoca al delegado. Cualquier método referenciado por este delegado se ejecutará en este punto.

11. Hacer click con el botón derecho en la carpeta References del proyecto DelegatesForms y hacer click en Agregar Referencia (Add Reference).

12. En el cuadro de diálogo Administrador de Referencias, en el panel de la izquierda hacer click en Solución (Solution). En el panel del centro seleccionar el proyecto CheckoutService y aceptar.

13. Regresar al archivo MainWindow.cs del proyecto DelegatesForms y agregar la siguiente directiva using al principio:

```
using CheckoutService;
```

14. Agregar la variable privada *checkoutController* de tipo *CheckoutController* en la clase MainWindow, como se muestra:

```
public ... class MainWindow : ...
{
   ...
   private Auditor _auditor = null;
   private Shipper _shipper = null;
   private CheckoutController _checkoutController = null;
   ...
}
```

15. Ubicar el constructor y luego de la creación de los componentes de *Auditor* y *Shipper* instanciar el componente *CheckoutController*, como sigue:

```
public MainWindow()
{
      ...
      _auditor = new Auditor();
      _shipper = new Shipper();
      _checkoutController = new CheckoutController();
}
```

16. Agregar las siguientes declaraciones, en negrita, en el constructor, debajo de la declaración del paso anterior:

```
_checkoutController.CheckoutProcessing += _auditor.AuditOrder;
_checkoutController.CheckoutProcessing += _shipper.ShipOrder;
```

Este código agrega referencias a los métodos *AuditOrder* y *ShipOrder*, de los objetos *_auditor* y *_shipper*, al delegado *CheckoutProcessing* del objeto *_checkoutController*.

17. Ubicar el método *CheckoutButtonClicked* en la clase MainWindow. Reemplazar el bloque if con lo siguiente:

```
try
{
      // Perform the checkout processing
      _checkoutController.StartCheckoutProcessing(_order);

      // Display a summary of the order
      ...
}
```

De esta manera se logró desacoplar la lógica de pago de los componentes que usan este proceso de pago. La lógica de negocios en la clase *MainWindow* especifica qué componentes que *CheckoutController* debe usar.

18. Probar nuevamente la aplicación.