

Lab Assignment 12

Course: CS202 Software Tools and Techniques for CSE

Lab Topic: Advanced Event Handling and Custom EventArgs in C# Windows Forms Applications

Date: 10th November 2025

Objective

This lab deepens understanding of advanced event-driven mechanisms in C# Windows Forms Applications. Students will design modular GUIs that demonstrate event chaining, filtered event invocation, and contextual data sharing through custom EventArgs.

Learning Outcomes

By the end of this lab, students will be able to:

- ✓ Implement custom EventArgs classes to exchange contextual data between publishers and subscribers.
- ✓ Demonstrate event chaining where one event dynamically triggers another.
- ✓ Apply conditional event firing and multicast subscription in GUI contexts.
- ✓ Design reusable event-driven components that decouple logic and user interface layers.

Lab Requirements

- Operating System: Windows
- Software: Visual Studio 2022 (Community Edition) with .NET SDK
- Programming Language: C# (latest stable version)

Lab Activities:

1. Windows Forms App – Multi-Stage Event Chaining with Custom EventArgs:

Develop a Windows Forms application named **OrderPipeline** that models a small order-processing workflow using chained custom events.

➤ Form Components:

1. TextBox: for entering **Customer Name**
2. ComboBox: with product options (“Laptop”, “Mouse”, “Keyboard”)
3. NumericUpDown: for **Quantity**
4. Button: **btnProcessOrder**
5. Label: **lblStatus**

➤ Functionality:

1. Clicking Process Order raises a custom event **OrderCreated** using

```
public class ShipEventArgs : EventArgs
{
    public string Product { get; }
    public bool Express { get; }
    public ShipEventArgs(string p, bool ex)
        => (Product, Express) = (p, ex);
}
```

2. **OrderCreated** has two subscribers:

- **ValidateOrder()** → Checks if **quantity > 0** and updates **lblStatus** with "Validated". If invalid, triggers another event **OrderRejected**.
- **DisplayOrderInfo()** → Shows **MessageBox** with order summary.

Note: Please reach out to the TAs for any queries/issues.

3. When **OrderRejected** is fired, its subscriber **ShowRejection()** updates the label with "**Order Invalid – Please retry**".
4. If order is valid, **ValidateOrder()** must chain another event **OrderConfirmed**, whose subscriber **ShowConfirmation()** changes label to "**Order Processed Successfully for <Customer>**".

2. Event Filtering and Dynamic Subscriber Management:

Extend Task 1 to show **event filtering** and **dynamic subscription**.

➤ **Modifications:**

1. Add a **CheckBox** named **chkExpress** and a new **Button** named **btnShipOrder..**
2. Define another event **OrderShipped** (similar to OrderEventArgs in Task 1) with a new **ShipEventArgs** class that includes **Product** and **Express** fields.

➤ **Subscribers for OrderShipped:**

1. **ShowDispatch()** → Displays "**Product dispatched: [Product]**" in the label.
2. **NotifyCourier()** → Displays a **MessageBox** with "**Express delivery initiated!**" if **Express == true**.

➤ Implement **event filtering** such that **NotifyCourier()** is **added as a subscriber** only when the user checks **chkExpress**.

If the checkbox is unchecked, remove this subscriber dynamically **before raising the event**.

➤ Show that pressing **Ship Order** triggers the event only if the previous order was confirmed (use a boolean flag to track status).

Demonstration Output:

Include screenshots of:

- Code showing delegate and event declarations.
- GUI state for both express and regular shipping.
- Dynamic subscriber addition/removal in action.

3. Output Reasoning (Level 0)

- **Does this code compile? If not, identify the error and explain why. If it does, what would print?**

```
public delegate void AuthCallback(bool validUser);
public static AuthCallback loginCallback = Login;
public static void Login()
{
    Console.WriteLine("Valid user!");
}

public static void Main(string[] args)
{
    loginCallback(true);
}
```

- **What will be the output of the following C# code? Why?**

```
using System;

delegate void Notify(string msg);
```

Note: Please reach out to the TAs for any queries/issues.

```

class Program
{
    static void Main()
    {
        Notify handler = null;

        handler += (m) => Console.WriteLine("A: " + m);
        handler += (m) => Console.WriteLine("B: " + m.ToUpper());

        handler("hello");

        handler -= (m) => Console.WriteLine("A: " + m);
        handler("world");
    }
}

```

4. Output Reasoning (Level 1)

- What will be the output of the following C# code? Why?

```

using System;

class Program
{
    static string txtAge;
    static DateTime selectedDate;
    static int parsedAge;

    static void Main(string[] args)
    {
        try
        {

            Console.WriteLine(txtAge == null ? "txtAge is null" : txtAge);

            Console.WriteLine(selectedDate == default(DateTime)
                ? "selectedDate is default"
                : selectedDate.ToString());

            if (string.IsNullOrEmpty(txtAge))
            {
                Console.WriteLine("txtAge is null or empty, cannot parse");
            }
            else
            {
                parsedAge = int.Parse(txtAge);
                Console.WriteLine($"Parsed Age: {parsedAge}");
            }
        }
        catch (FormatException)
        {
            Console.WriteLine("Format Exception Caught");
        }
        catch (ArgumentNullException)
        {
            Console.WriteLine("ArgumentNullException Caught");
        }
        finally
        {
            Console.WriteLine("Finally block executed");
        }
    }
}

```

- What will be the output of the following C# code? Why?

```

using System;

```

Note: Please reach out to the TAs for any queries/issues.

```

delegate void Operation();

class Program
{
    static void Main()
    {
        Operation ops = null;

        ops += Step1;
        ops += Step2;
        ops += Step3;

        try
        {
            ops();
        }
        catch (Exception ex)
        {
            Console.WriteLine("Caught: " + ex.Message);
        }

        Console.WriteLine("End of Main");
    }

    static void Step1()
    {
        Console.WriteLine("Step 1");
    }

    static void Step2()
    {
        Console.WriteLine("Step 2");
        throw new InvalidOperationException("Step 2 failed!");
    }

    static void Step3()
    {
        Console.WriteLine("Step 3");
    }
}

```

5. Output Reasoning (Level 2)

- What will be the output of the following C# code? Why?

```

using System;

namespace MethodOverloadingExample
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 5;
            new Base().F(x);
            new Derived().F(x);

            Console.ReadKey();
        }
    }

    class Base
    {
        public void F(int x)
        {
            Console.WriteLine("Base.F(int)");
        }
    }
}

```

Note: Please reach out to the TAs for any queries/issues.

```

class Derived : Base
{
    public void F(double x)
    {
        Console.WriteLine("Derived.F(double)");
    }
}

```

➤ What will be the output of the following C# code? Why?

```

using System;

class StepEventArgs : EventArgs
{
    public int Step { get; }
    public StepEventArgs(int s) => Step = s;
}

class Workflow
{
    public event EventHandler<StepEventArgs> StepStarted;
    public event EventHandler<StepEventArgs> StepCompleted;

    public void Run()
    {
        for (int i = 1; i <= 3; i++)
        {
            StepStarted?.Invoke(this, new StepEventArgs(i));
            Console.WriteLine($"[{i}]");
            StepCompleted?.Invoke(this, new StepEventArgs(i));
        }
    }
}

class Program
{
    static void Main()
    {
        Workflow wf = new Workflow();

        wf.StepStarted += (s, e) =>
        {
            Console.WriteLine("<S" + e.Step + ">");
            if (e.Step == 2)
                ((Workflow)s).StepCompleted += (snd, ev)
                    => Console.WriteLine("(Dyn" + ev.Step + ")");
        };
        wf.StepCompleted += (s, e) => Console.WriteLine("<C" + e.Step + ">");

        wf.Run();
    }
}

```

Resources

- [Object.Equals Method](#)
- [Lecture 13](#)
- [Lecture 12](#)
- [Introduction to delegates and events in C#](#)
- [How to combine delegates \(Multicast Delegates\)\]](#)

Note: Please reach out to the TAs for any queries/issues.