Lab Assignment 11

Course: CS202 Software Tools and Techniques for CSE
Lab Topic: Events and Delegates in C# Windows Forms Applications
Date: 3rd November 2025

Objective
This lab deepens understanding of event handling and delegates in C# Windows Forms Applications. Students will learn how to create, subscribe, and invoke custom events to achieve modular, interactive, and reusable GUI designs using Visual Studio.

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

- ✓ Implement and handle custom events in C# Windows Forms applications using the publisher–subscriber model.
- ✓ Design and manage interactive form controls that respond dynamically to user actions through event-driven logic.
- ✓ Apply multicast event handling to connect multiple subscribers to a single event in a GUI application.
- ✓ Use custom **EventArgs** classes to pass contextual information between event publishers and subscribers.

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-Control Event Interaction:**
   Develop a Windows Forms application named `EventPlayground` to demonstrate event handling between multiple controls.
   - ➢ Form Components:
     1. Two Buttons: `btnChangeColor` and `btnChangeText`
     2. One Label: initially displaying "**Welcome to Events Lab**"
     3. A `ComboBox` with colour options (**Red, Green, Blue**)
   - ➢ Functionality:
     1. Clicking `btnChangeColor` triggers a user-defined event `ColorChangedEvent` that updates the Label's foreground colour based on the `ComboBox` selection.
     2. Clicking `btnChangeText` raises another custom event `TextChangedEvent`, which updates the Label text dynamically to show the current date and time.
   - ➢ Ensure events are declared and invoked using **custom delegates** (not only built-in Click events).
2. **Using EventArgs and Multiple Subscribers:**
   Extend Task 1 by defining a **custom EventArgs** class named `ColorEventArgs` that holds the selected color name.
   - ➢ Modify `ColorChangedEvent` to pass this data to multiple subscribers:

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

1. `UpdateLabelColor()` – changes label colour.
2. `ShowNotification()` – displays a message box showing the selected colour.

➢ Demonstrate how the **same** event can invoke multiple methods **(multicast behaviour).**

Important

- Use custom events and delegates in Windows Forms applications instead of relying solely on default control events.
- All input and output must occur through the graphical interface — no console interaction is required.
- Submit screenshots of: Source code (with delegate/event declarations visible), GUI showing correct event responses.

3. Output Reasoning (Level 0)

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

```
using System;


delegate int Calc(int x, int y);


class Program
{
    static int Add(int a, int b) { Console.Write("A"); return a + b; }
    static int Mul(int a, int b) { Console.Write("M"); return a * b; }
    static int Sub(int a, int b) { Console.Write("S"); return a - b; }


    static void Main()
    {
        Calc c = Add;
        c += Mul;
        c += Sub;
        c -= Add;
        int res = c(2, 3);
        Console.Write(":" + res);
    }
}
```

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

```
using System;

delegate void ActionHandler(ref int x);

class Program
{
    static void Inc(ref int a) { a += 2; Console.Write("I" + a + " "); }
    static void Dec(ref int a) { a--; Console.Write("D" + a + " "); }

    static void Main()
    {
        int val = 3;
        ActionHandler act = Inc;
        act += Dec;
        act(ref val);
        Console.Write("F" + val);
    }
}
```

4. Output Reasoning (Level 1)

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

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

```
using System;

class LimitEventArgs : EventArgs
{
    public int CurrentValue { get; }
    public LimitEventArgs(int val) => CurrentValue = val;
}

class Counter
{
    public event EventHandler<LimitEventArgs> LimitReached;
    public event EventHandler<LimitEventArgs> MilestoneReached;

    private int value = 0;

    public void Increment()
    {
        value++;
        Console.Write(">" + value);

        // Fire Milestone event every 2nd increment
        if (value % 2 == 0)
            MilestoneReached?.Invoke(this, new LimitEventArgs(value));

        // Fire Limit event every 3rd increment
        if (value % 3 == 0)
            LimitReached?.Invoke(this, new LimitEventArgs(value));
    }
}

class Program
{
    static void Main()
    {
        Counter c = new Counter();

        // Subscribers for LimitReached
        c.LimitReached += (s, e) => Console.Write("[L" + e.CurrentValue + "]");
        c.LimitReached += (s, e) => Console.Write("(Reset)");

        // Subscribers for MilestoneReached
        c.MilestoneReached += (s, e) =>
        {
            Console.Write("[M" + e.CurrentValue + "]");
            if (e.CurrentValue == 4)
                Console.Write("{Alert}");
        };

        for (int i = 0; i < 6; i++)
            c.Increment();
    }
}
```

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

```
using System;

class TemperatureEventArgs : EventArgs
{
    public int OldTemperature { get; }
    public int NewTemperature { get; }

    public TemperatureEventArgs(int oldTemp, int newTemp)
    {
        OldTemperature = oldTemp;
        NewTemperature = newTemp;
    }
}

class TemperatureSensor
{
    public event EventHandler<TemperatureEventArgs> TemperatureChanged;

    private int temperature = 25;

    public void UpdateTemperature(int newTemp)
    {
        int oldTemp = temperature;
        temperature = newTemp;
```

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

```
        if (Math.Abs(newTemp - oldTemp) > 5)
        {
            TemperatureChanged?.Invoke(this, new TemperatureEventArgs(oldTemp, newTemp));
        }
    }
}

class Program
{
    static void Main()
    {
        TemperatureSensor sensor = new TemperatureSensor();

        sensor.TemperatureChanged += (s, e) =>
            Console.WriteLine($"Temperature changed from {e.OldTemperature}°C to {e.NewTemperature}°C");

        sensor.TemperatureChanged += (s, e) =>
        {
            if (Math.Abs(e.NewTemperature - e.OldTemperature) > 10)
                Console.WriteLine(" Warning: Sudden change detected!");
        };

        sensor.UpdateTemperature(28);
        sensor.UpdateTemperature(30);
        sensor.UpdateTemperature(46);
        sensor.UpdateTemperature(52);
    }
}
```

5. Output Reasoning (Level 2)

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

```
using System;

class NotifyEventArgs : EventArgs
{
    public string Message { get; }
    public NotifyEventArgs(string msg) => Message = msg;
}

class Notifier
{
    public event EventHandler<NotifyEventArgs> OnNotify;

    public void Trigger(string msg)
    {
        Console.Write("[Start]");
        OnNotify?.Invoke(this, new NotifyEventArgs(msg));
        Console.Write("[End]");
    }
}

class Program
{
    static void Main()
    {
        Notifier n = new Notifier();

        n.OnNotify += (s, e) =>
        {
            Console.Write("{" + e.Message + "}");
        };

        n.OnNotify += (s, e) =>
        {
            Console.Write("(Nested)");
            if (e.Message == "Ping")
                ((Notifier)s).Trigger("Pong");
        };

        n.Trigger("Ping");
    }
}
```

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

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

```
using System;

class AlertEventArgs : EventArgs
{
    public string Info { get; }
    public AlertEventArgs(string info) => Info = info;
}

class Sensor
{
    public event EventHandler<AlertEventArgs> ThresholdReached;

    public void Check(int value)
    {
        Console.Write("[Check]");
        if (value > 50)
            ThresholdReached?.Invoke(this, new AlertEventArgs("High"));
        Console.Write("[Done]");
    }
}

class Program
{
    static void Main()
    {
        Sensor s = new Sensor();

        s.ThresholdReached += (sender, e) =>
        {
            Console.Write("{" + e.Info + "}");
            if (e.Info == "High")
                ((Sensor)sender).Check(30);
        };

        s.ThresholdReached += (sender, e) =>
            Console.Write("(Alert)");

        s.Check(80);
    }
}
```

Resources

- Lecture 12 Slides
- Microsoft Learn – Delegates and Events in C#
- C# Delegates Overview
- Windows Forms Events Guide
- Event-driven Programming Concepts

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