Lab Report 12: Event-driven Programming with Windows Forms in C#

Kushal Rathod
22110128
CS202: Software Tools and Techniques for CSE

April 2025

1 Introduction

This lab exercise explores the event-driven programming model in C#, focusing on building applications that respond to user-generated and system-generated events. Using Visual Studio 2022 and the .NET platform, we create both console-based and Windows Forms applications that demonstrate the use of events, delegates, and user interfaces.

Students are introduced to key concepts such as the publisher-subscriber model, user-defined events, event handlers, and background color transitions in forms. The objective is to understand how C# applications can react dynamically based on runtime conditions and user inputs.

1.1 Learning Objectives

The lab focuses on:

- Developing Windows Forms Applications using C#
- Understanding the event-driven programming paradigm
- Working with user-defined events using the publisher-subscriber model
- Building GUI-based applications with responsive elements
- Handling time-based state changes and form updates

1.2 Development Environment

Tools Used:

- OS: Windows 11
- IDE: Visual Studio 2022 (Community Edition)
- Framework: .NET 6.0
- Language: C# 10.0

2 Implementation Approach

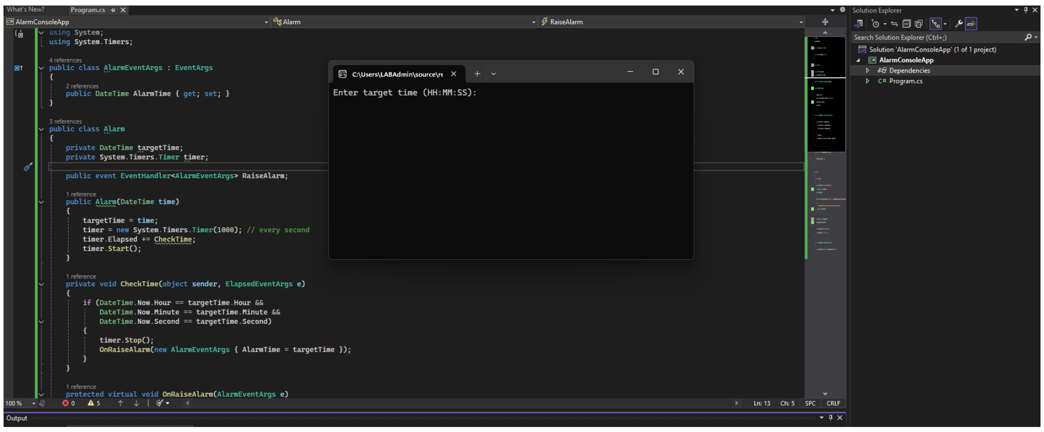
2.1 Exercise 1: Console Application with Custom Event - complete code in Github repo

The goal of this exercise was to create a console application where the user enters a target time in `HH:MM:SS` format. The program continuously compares this with the system time. When they match, a custom event `raiseAlarm` is triggered, which in turn invokes the `Ring_alarm()` function to display a message.

Key Concepts Used:

- Delegate and event declaration
- Time comparison using `DateTime.Now`
- Subscriber/Publisher model
- Thread sleep mechanism to poll every second

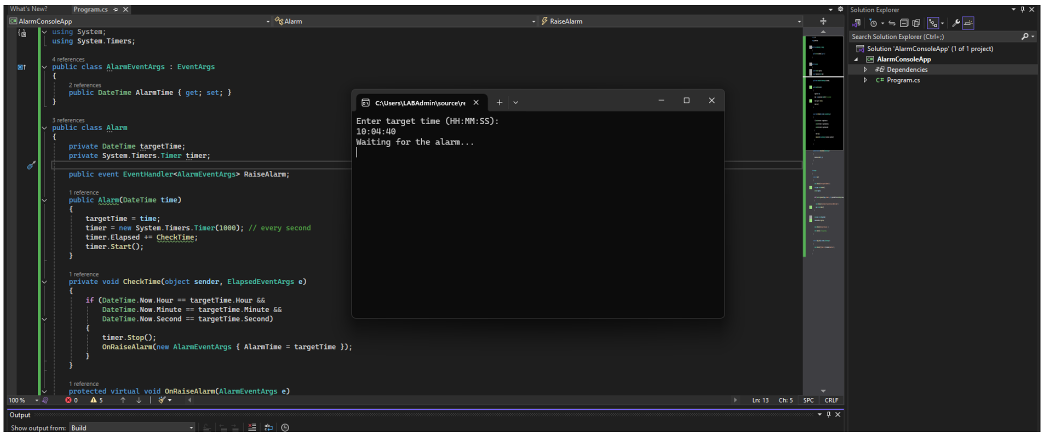
Note: Screenshots of the full console code and output are attached below.



```
using System;
using System.Timers;

public class AlarmEventArgs : EventArgs
{
    public DateTime AlarmTime { get; set; }
}

public class Alarm
{
    private DateTime targetTime;
    private System.Timers.Timer timer;

    public event EventHandler<AlarmEventArgs> RaiseAlarm;

    public Alarm(DateTime time)
    {
        targetTime = time;
        timer = new System.Timers.Timer(1000); // every second
        timer.Elapsed += CheckTime;
        timer.Start();
    }

    private void CheckTime(object sender, ElapsedEventArgs e)
    {
        if (DateTime.Now.Hour == targetTime.Hour &&
            DateTime.Now.Minute == targetTime.Minute &&
            DateTime.Now.Second == targetTime.Second)
        {
            timer.Stop();
            OnRaiseAlarm(new AlarmEventArgs { AlarmTime = targetTime });
        }
    }

    protected virtual void OnRaiseAlarm(AlarmEventArgs e)
```

```
Enter target time (HH:MM:SS):
```

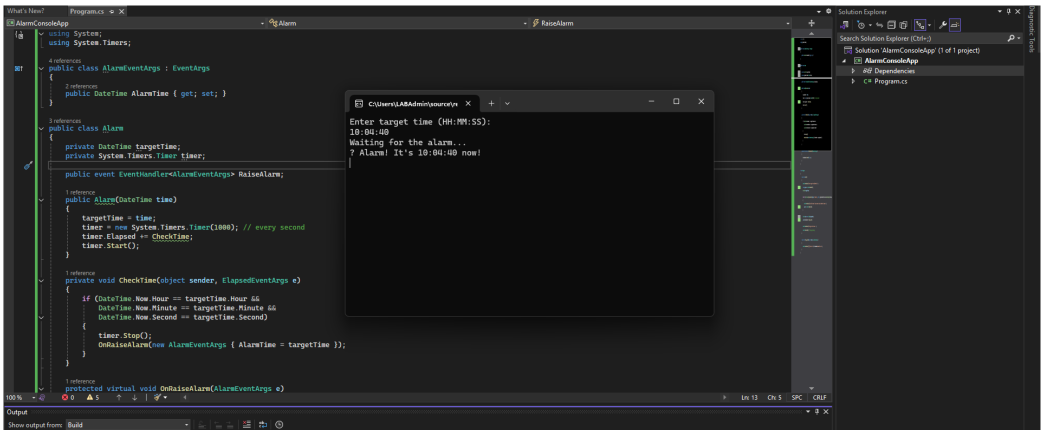
Figure 1: This is the ready state of the console app where it is ready to take time input for alarm



```
using System;
using System.Timers;

public class AlarmEventArgs : EventArgs
{
    public DateTime AlarmTime { get; set; }
}

public class Alarm
{
    private DateTime targetTime;
    private System.Timers.Timer timer;

    public event EventHandler<AlarmEventArgs> RaiseAlarm;

    public Alarm(DateTime time)
    {
        targetTime = time;
        timer = new System.Timers.Timer(1000); // every second
        timer.Elapsed += CheckTime;
        timer.Start();
    }

    private void CheckTime(object sender, ElapsedEventArgs e)
    {
        if (DateTime.Now.Hour == targetTime.Hour &&
            DateTime.Now.Minute == targetTime.Minute &&
            DateTime.Now.Second == targetTime.Second)
        {
            timer.Stop();
            OnRaiseAlarm(new AlarmEventArgs { AlarmTime = targetTime });
        }
    }

    protected virtual void OnRaiseAlarm(AlarmEventArgs e)
```

```
Enter target time (HH:MM:SS):
10:04:40
Waiting for the alarm...
```

Figure 2: This is the waiting state of the console app where it is waiting for the system time to match with the alarm input time



```
using System;
using System.Timers;

4 references
public class AlarmEventArgs : EventArgs
{
    2 references
    public DateTime AlarmTime { get; set; }
}

3 references
public class Alarm
{
    private DateTime targetTime;
    private System.Timers.Timer timer;

    public event EventHandler<AlarmEventArgs> RaiseAlarm;

    1 reference
    public Alarm(DateTime time)
    {
        targetTime = time;
        timer = new System.Timers.Timer(1000); // every second
        timer.Elapsed += CheckTime;
        timer.Start();
    }

    1 reference
    private void CheckTime(object sender, ElapsedEventArgs e)
    {
        if (DateTime.Now.Hour == targetTime.Hour &&
            DateTime.Now.Minute == targetTime.Minute &&
            DateTime.Now.Second == targetTime.Second)
        {
            timer.Stop();
            OnRaiseAlarm(new AlarmEventArgs { AlarmTime = targetTime });
        }
    }

    1 reference
    protected virtual void OnRaiseAlarm(AlarmEventArgs e)
```

```
C:\Users\LABAdmin\source\re
Enter target time (HH:MM:SS):
10:04:40
Waiting for the alarm...
? Alarm! It's 10:04:40 now!
```

Figure 3: This is the alarm state of the console app where the system time has matched the alarm time and the alarm has rung

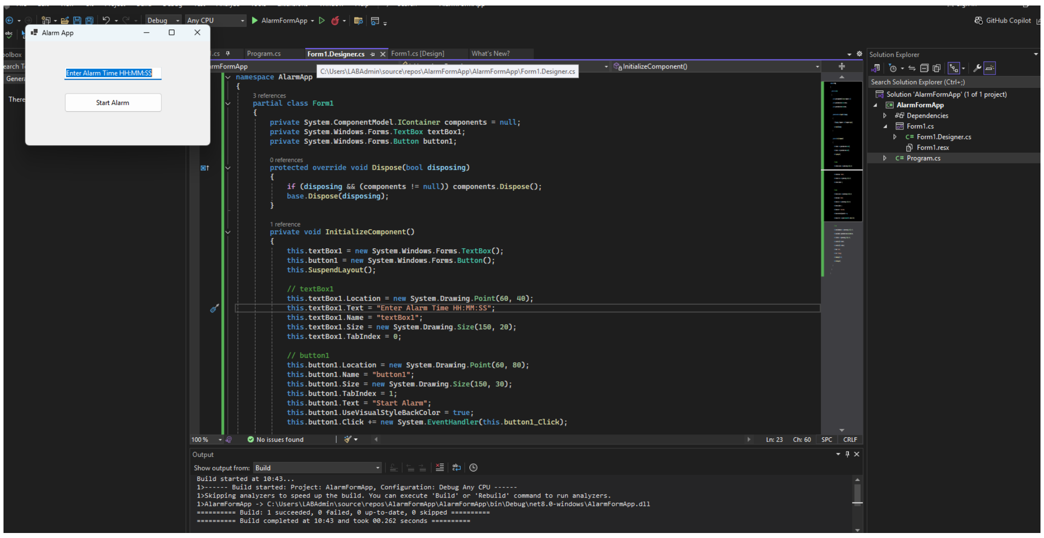
2.2 Exercise 2: Windows Forms Application (GUI) - complete code and video demonstration in Github repo

In this task, the previous console app is converted to a Windows Forms application. The user enters time into a textbox and clicks "Start". Every second, the form's background color changes. When the target time matches the system time, color transition stops and a message box displays the alert.

Main Components:

- TextBox (for input)
- Button (to start alarm)
- Timer control (set to tick every second)
- Event handler for Timer and Button click
- `MessageBox.Show()` for alarm message

Note: Screenshots showing the Form design and the C# event-handling code are included below.



```csharp
namespace AlarmApp
{
    partial class Form1
    {
        private System.ComponentModel.IContainer components = null;
        private System.Windows.Forms.TextBox textBox1;
        private System.Windows.Forms.Button button1;

        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null)) components.Dispose();
            base.Dispose(disposing);
        }

        private void InitializeComponent()
        {
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.button1 = new System.Windows.Forms.Button();
            this.SuspendLayout();

            // textBox1
            this.textBox1.Location = new System.Drawing.Point(60, 40);
            this.textBox1.Text = "Enter Alarm Time HH:MM:SS";
            this.textBox1.Name = "textBox1";
            this.textBox1.Size = new System.Drawing.Size(150, 20);
            this.textBox1.TabIndex = 0;

            // button1
            this.button1.Location = new System.Drawing.Point(60, 80);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(150, 30);
            this.button1.TabIndex = 1;
            this.button1.Text = "Start Alarm";
            this.button1.UseVisualStyleBackColor = true;
            this.button1.Click += new System.EventHandler(this.button1_Click);
```

```
Build started at 10:43...
1>------ Build started: Project: AlarmFormApp, Configuration: Debug Any CPU ------
1>Skipping analyzers to speed up the build. You can execute 'Build' or 'Rebuild' command to run analyzers.
1>AlarmFormApp -> C:\Users\LABAdmin\source\repos\AlarmFormApp\AlarmFormApp\bin\Debug\net8.0-windows\AlarmFormApp.dll
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Build completed at 10:43 and took 00.262 seconds ==========
```

Figure 4: Windows Forms app GUI for time input and alarm trigger

3 Results and Observations

- The console application correctly raises a custom event and displays a message when the target time is reached. The polling was set with `Thread.Sleep(1000)` for efficiency.
- In the Windows Forms application, the background color changes using a predefined set of colors. The change halts when the entered time matches the system clock, and a pop-up confirms alarm activation.
- The usage of the Timer component allowed accurate tick events without consuming unnecessary CPU cycles.
- Input validation was assumed to be correct as per instructions, but boundary cases (e.g., malformed inputs) can be handled using `DateTime.TryParse()`.

Discussion and Conclusion

Challenges Faced

- Implementing the custom event system in a console application required a solid understanding of C# delegates and events.

- Synchronizing the Windows Forms Timer with real-time clock comparisons introduced minor timing mismatches during testing.
- Handling GUI updates without freezing the UI thread was critical and required cautious usage of event handlers.

Reflections

This lab highlighted how reactive programming works in a practical GUI environment. It also demonstrated the difference between polling (in console apps) versus event-based ticks (in forms), helping understand the performance and responsiveness trade-offs.

Lessons Learned

1. **Understand Event Flow:** Properly wiring events and delegates helps build modular and reusable logic.
2. **UI Programming Is Event-Driven:** Forms depend entirely on events like clicks and ticks; designing for responsiveness is crucial.
3. **Thread.Sleep vs. Timer:** For UI apps, Timer is the right approach — `Thread.Sleep()` can freeze the form.
4. **Always Validate Input:** Even when validation is skipped, be cautious with input format to prevent runtime errors.

Summary

Lab 12 gave an insightful tour into event-driven programming. Starting with a terminal-based time tracker that triggered an alarm, and culminating with an interactive form that dynamically responded to time updates, this lab demonstrated how to work with C# events and GUI development using Visual Studio.

Note: **All code, screenshots and demonstration videos are submitted in the GitHub repository under the branch `STT_lab12`. Demo videos (.mkv) showing alarm triggers in app2 is included in the media folder.**