NAVIGATING COMMON C# CODE PITFALLS:

TIPS AND BEST PRACTICES FOR DEVELOPER SUCCESS

Nadee Kodituwakku

#1 Overusing var keyword

- “var” ≠ variant

> It is important to understand that the `var` keyword does not mean "variant" and does not indicate that the variable is loosely typed, or late-bound. It just means that the compiler determines and assigns the most appropriate type.

Source: Implicitly typed local variables - C# | Microsoft Learn

- Compiler infer the type based on the initialization expression.
- Used to declare implicitly-typed local variables.

#1 Overusing var keyword

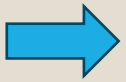
Method return types:

```
var result = ProcessOrderDisplay();
```

```
int result = ProcessOrderDisplay();
```

Unclear variable types in loops:

```
foreach (var item in someList)
{
    item.IsMissing();
    item.MarkAsRetired();
}
```



```
foreach (Car item in someList)
{
    item.IsMissing();
    item.MarkAsRetired();
}
```

👎 Lacks readability

👍 Improves readability

When to use var???

*** When the right-hand side of the assignment is clear:**

```
var person = new Person();
```

*** Syntactic convenience**:

```
IEnumerable<string> query = from p in people
                            where p.Age > 30
                            select p.Name;
```

```
var query = from p in people
            where p.Age > 30
            select p.Name;
```

👎 Noisy code

👎 Doesn't add much clarity

👍 Cleaner code

👍 Type is understood based on the link query

When to use var???

* **Anonymous types :**

```
string[] words = { "aPPLE", "BlUeBeRrY", "cHeRry" };

// If a query produces a sequence of anonymous types,
// then use var in the foreach statement to access the properties.
var upperLowerWords =
    from w in words
    select new { Upper = w.ToUpper(), Lower = w.ToLower() };
```

```
// Execute the query
foreach (var ul in upperLowerWords)
{
    Console.WriteLine("Uppercase: {0}, Lowercase: {1}", ul.Upper, ul.Lower);
}
```

#2 Events

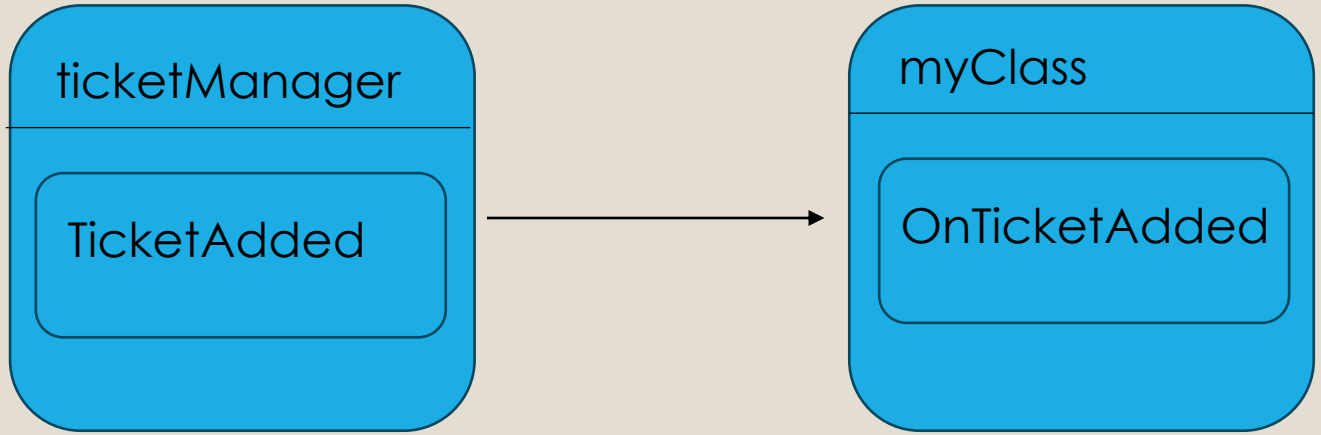
```
2 references
public class MyClass
{
    1 reference
    public MyClass(TicketManager ticketManager)
    {
        ticketManager.TicketAdded += OnTicketAdded;
    }

    1 reference
    private void OnTicketAdded(object sender, TicketEventArgs e)
    {
        // someone clicked me?
    }

    0 references
    public void DoSomething()
    {
        // do something
    }
}
```

Assume **SomeMethod()** is executed once and never used again.

```
0 references
public void SomeMethod(TicketManager ticketManager)
{
    var myClass = new MyClass(ticketManager);
    myClass.DoSomething();
}
```

```
{
    ticketManager.TicketAdded += OnTicketAdded;
}
```

- When myClass subscribe to the event provided by ticketManager, ticketManager obtains a reference to myClass.



- **Understanding the Problem**: Event subscriptions can cause memory leaks if the event publisher outlives the subscriber. Any instance of MyClass that is referenced by ticketManager will never be collected by GC .

How to fix this?

💡 **Unsubscribe from Events**:

Always unsubscribe from events when the subscriber is no longer needed to prevent memory leaks.

```
ticketManager.TicketAdded -= OnTicketAdded;
```

💡 **Weak Event Pattern**: Use weak event patterns to allow the garbage collector to collect subscribers even if they are still subscribed.

Four ways of implementing weak events:

1. Existing WeakEventManager class in WPF:

```
LostFocusEventManager.AddHandler(source, Source_LostFocus);
```

2. Generic weak event manager WeakEventManager<TEventSource,TEventArgs>:

```
WeakEventManager<SomeEventSource, SomeEventArgs>.AddHandler(source, "SomeEvent", Source_SomeEvent);
```

3. Custom weak event manager

4. Third-party weak event manager

💡 Subscribe with Anonymous methods without capturing any members:

- EventSubscriber class subscribes to the EventPublisher's OnMessageReceived event using a lambda expression.

Lamda Expression

- No variables captured from the surrounding context.

```
2 references
public class EventPublisher
{
    // Define an event
    public event Action<string> OnMessageReceived;

    1 reference
    public void PublishMessage(string message)
    {
        // Raise the event
        OnMessageReceived?.Invoke(message);
    }
}

1 reference
public class EventSubscriber
{
    1 reference
    public void Subscribe(EventPublisher publisher)
    {
        // Subscribe using an anonymous function (lambda expression)
        publisher.OnMessageReceived += (message) => Console.WriteLine($"Received message: {message}");
    }
}

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        var publisher = new EventPublisher();
        var subscriber = new EventSubscriber();

        // Subscriber subscribes to the publisher's event
        subscriber.Subscribe(publisher);

        // Publisher publishes a message
        publisher.PublishMessage("Hello, World!");
    }
}
```

#3 Any or Count or Length ??

```
IEnumerable<string> collection = new List<string>();

if (collection != null && collection.Any())
{
    // Collection is not null and has elements
}
```

- Depends on the type of collection you're working with
- **IEnumerable<T> => Any():**
  - Faster than Count()
  - Any() - stops as soon as it finds an element
  - Count() - iterates through all the elements

#3 Any or Count or Length ??

```
List<int> list = new List<int>();

if (list != null && list.Count > 0)
{
    // Collection is not null and has elements
}
```

- **List<T>, Array, ICollection<T>, ICollection => Count**
  - Knows its size
  - Property is already available and do not require any iteration.

#3 Any or Count or Length ??

```
int[] intArray = new int[12];

if (intArray != null && intArray.Length > 0)
{
    // Array is not null and has elements
}
```

- **Array => Length**
  - Specific to Arrays
  - Returns the size of the array

#4 using statement

- Ensure the correct use of disposable objects

Let’s do a code review on a real-world issue.

```
119
120             using (Bitmap tempBmp = (Bitmap)Image.FromFile(Path.Combine(filePath, imageArray[0])))
121             {
122                 using (Graphics graphics = Graphics.FromImage(tempBmp))
123                 {
124                     Bitmap newLayer = null;
125                     for (int i = 0; i < imageArray.Length; i++)
126                     {
127                         newLayer = (Bitmap)Image.FromFile(Path.Combine(filePath, imageArray[i]));
128                         graphics.DrawImage(newLayer, 0, 0);
129                     }
130
131                     if (!reducedText)
132                     {
133                         PrivateFontCollection collection = new PrivateFontCollection();
134                         collection.AddFontFile(Path.Combine(fontPath, "arial.ttf"));
135                         collection.AddFontFile(Path.Combine(fontPath, "arialbd.ttf"));
136                         Font fontRegular = new Font(collection.Families.First(), 18, FontStyle.Regular, GraphicsUnit.Pixel);
137                         Font fontBold = new Font(collection.Families.First(), 42, FontStyle.Bold, GraphicsUnit.Pixel);
138                         SolidBrush blackBrush = new SolidBrush(Color.Black);
139                         StringFormat sf = new StringFormat();
140                         sf.Alignment = StringAlignment.Center;
141
142                         graphics.DrawString(imageTitle1, fontBold, blackBrush, 730, 50, sf);
143
144                         graphics.DrawString(imageTitle2, fontBold, blackBrush, 730, 100, sf);
145
146                         graphics.DrawString(imageTitleDate, fontBold, blackBrush, 730, 150, sf);
147
148                         graphics.DrawString(imageForecastDate, fontRegular, blackBrush, 220, 900, sf);
149                     }
150
151                     graphics.TextRenderingHint = TextRenderingHint.AntiAliasGridFit;
152                     graphics.SmoothingMode = SmoothingMode.AntiAlias;
153                 }
154
155                 using (var memoryStream = new MemoryStream())
156                 {
157                     tempBmp.Save(memoryStream, System.Drawing.Imaging.ImageFormat.Png);
158                     return memoryStream.ToArray();
159                 }
160             }
161         }
162
```

```
                    , northCentralFileName
                    , southWestFileName
                };

            using (Bitmap tempBmp = (Bitmap)Image.FromFile(Path.Combine(filePath, imageArray[0])))
            {
                using (Graphics graphics = Graphics.FromImage(tempBmp))
                {
                    Bitmap newLayer = null;
                    for (int i = 0; i < imageArray.Length; i++)
                    {
                        newLayer = (Bitmap)Image.FromFile(Path.Combine(filePath, imageArray[i]));
                        graphics.DrawImage(newLayer, 0, 0);
                    }

                    if (!reducedText)
                    {
                        PrivateFontCollection collection = new PrivateFontCollection();
                        collection.AddFontFile(Path.Combine(fontPath, "arial.ttf"));
                        collection.AddFontFile(Path.Combine(fontPath, "arialbd.ttf"));
                        Font fontRegular = new Font(collection.Families.First(), 18, FontStyle.Regular, GraphicsUnit.Pixel);
                        Font fontBold = new Font(collection.Families.First(), 42, FontStyle.Bold, GraphicsUnit.Pixel);
                        SolidBrush blackBrush = new SolidBrush(Color.Black);
                        StringFormat sf = new StringFormat();
                        sf.Alignment = StringAlignment.Center;

                        graphics.DrawString(imageTitle1, fontBold, blackBrush, 730, 50, sf);

                        graphics.DrawString(imageTitle2, fontBold, blackBrush, 730, 100, sf);

                        graphics.DrawString(imageTitleDate, fontBold, blackBrush, 730, 150, sf);

                        graphics.DrawString(imageForecastDate, fontRegular, blackBrush, 220, 900, sf);
                    }

                    graphics.TextRenderingHint = TextRenderingHint.AntiAliasGridFit;
                    graphics.SmoothingMode = SmoothingMode.AntiAlias;
                }

                using (var memoryStream = new MemoryStream())
                {
                    tempBmp.Save(memoryStream, System.Drawing.Imaging.ImageFormat.Png);
                    return memoryStream.ToArray();
                }
            }
        }
```

```
                    , northCentralFileName
                    , southWestFileName
                };

            using Bitmap tempBmp = (Bitmap)Image.FromFile(Path.Combine(filePath, imageArray[0]));

            using (Graphics graphics = Graphics.FromImage(tempBmp))
            {
                for (int i = 0; i < imageArray.Length; i++)
                {
                    using var newLayer = (Bitmap)Image.FromFile(Path.Combine(filePath, imageArray[i]));
                    graphics.DrawImage(newLayer, 0, 0);
                }

                if (!reducedText)
                {
                    using PrivateFontCollection collection = new PrivateFontCollection();
                    collection.AddFontFile(Path.Combine(fontPath, "arial.ttf"));
                    collection.AddFontFile(Path.Combine(fontPath, "arialbd.ttf"));
                    using Font fontRegular = new Font(collection.Families.First(), 18, FontStyle.Regular, GraphicsUnit.Pixel);
                    using Font fontBold = new Font(collection.Families.First(), 42, FontStyle.Bold, GraphicsUnit.Pixel);
                    using SolidBrush blackBrush = new SolidBrush(Color.Black);
                    using StringFormat sf = new StringFormat();
                    sf.Alignment = StringAlignment.Center;

                    graphics.DrawString(imageTitle1, fontBold, blackBrush, 730, 50, sf);

                    graphics.DrawString(imageTitle2, fontBold, blackBrush, 730, 100, sf);

                    graphics.DrawString(imageTitleDate, fontBold, blackBrush, 730, 150, sf);

                    graphics.DrawString(imageForecastDate, fontRegular, blackBrush, 220, 900, sf);
                }

                graphics.TextRenderingHint = TextRenderingHint.AntiAliasGridFit;
                graphics.SmoothingMode = SmoothingMode.AntiAlias;
            }

            using (var memoryStream = new MemoryStream())
            {
                tempBmp.Save(memoryStream, System.Drawing.Imaging.ImageFormat.Png);
                return memoryStream.ToArray();
            }
        }
```

Summary

#1 Overusing var keyword

#2 Events

#3 Any or Count or Length ??

#4 using statement

Thank you!!

Questions??