

# Events

---



**Filip Ekberg**

Principal Consultant & CEO

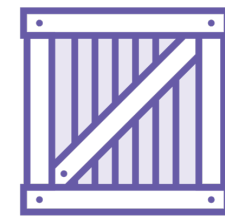
@fekberg   fekberg.com



# Broadcasting Events



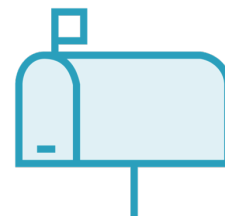
**Order Processor**



Order ready for shipment



Payment processed



Shipping label ready



**Subscribers of the events**



OrderNumber	IsReadyForShipment	Total
8101e88e-ff6a-4eb7-af07-bdaef0b711e6	<input checked="" type="checkbox"/>	0
0b7c7c69-c46a-4585-a3be-e4f3f0fe96d1	<input checked="" type="checkbox"/>	0

Run a method when this button is clicked



```
0 references
26 public MainWindow()
27 {
28     InitializeComponent();
29
30     ProcessOrder.Click += ProcessOrder_Click;
31 }
32
1 reference
33 private void ProcessOrder_Click(object sender,
34     RoutedEventArgs e)
35 {
36
37 }
38
```

# Invoking a Publicly Exposed Delegate

```
class OrderProcessor
{
    public Func<Order, bool> OnOrderInitialized { get; set; }
}
```

```
var processor = new OrderProcessor();
```

```
processor.OnOrderInitialized();
```



# Event

“The **event** keyword is used to **declare** an **event** in a **publisher class**.”

**Example:**

```
public event EventHandler OrderCreated;
```



# Event

“The **event** keyword is used to **declare** an **event** in a **publisher class**.”

**Example:**

```
public event EventHandler OrderCreated;
```

**Delegate**



# Events in .NET

## Publisher

The class that owns the event and is in charge of raising it

## Subscriber

The classes that subscribe to events exposed by the publisher





# Creating an **Event**

```
class OrderProcessor  
{  
  
}
```



# Creating an **Event**

```
class OrderProcessor  
{  
    public event  
}
```



# Creating an **Event**

```
class OrderProcessor
{
    public event EventHandler OrderCreated;
}
```



# Choosing a **Delegate** for the **Event**

**Don't use** your **own delegates**,  
**Func** or **Action**

An event delegate is **always**  
**void** and an event **never**  
**returns a value**



# Creating an **Event**

```
class OrderProcessor
{
    public event EventHandler OrderCreated;
}
```



Use the delegates **EventHandler** or **EventHandler<T>**



# EventHandler

## Sender

The instance of the publisher that raised the event

## EventArgs

Inherit from EventArgs to create a class that represents the event data



When you **add** the **event** keyword before a **delegate**, it can **only** be **invoked** from **that class**



# Accessing the **Event**

```
class OrderProcessor  
{  
    public event EventHandler OrderCreated;  
}
```

```
var processor = new OrderProcessor();
```

```
processor.OrderCreated
```





# Accessing the **Event**

```
class OrderProcessor
{
    public event EventHandler OrderCreated;
}

var processor = new OrderProcessor();

processor.OrderCreated = null; // Compiler error
```

 EventHandler OrderProcessor.OrderCreated

CS0070: The event 'OrderProcessor.OrderCreated' can only appear on the left hand side of += or -= (except when used from within the type 'OrderProcessor')



# Accessing the **Event**

```
class OrderProcessor
{
    public event EventHandler OrderCreated;
}

var processor = new OrderProcessor();

processor.OrderCreated += Processor_OrderCreated;
```



The **publisher** should **never**  
**care** about **what** the  
**subscriber** **does** when the  
event is raised



# Subscribers Are Executed **Sequentially**



## One slow subscriber?

This makes the whole method slow and impacts all the subscribers in the chain!



# Always Unsubscribe!

```
class OrderProcessor
{
    public event EventHandler OrderCreated;
}

var processor = new OrderProcessor();

processor.OrderCreated += Processor_OrderCreated;

...

processor.OrderCreated -= Processor_OrderCreated;
```



# Always Unsubscribe!

```
class OrderProcessor
{
    public event EventHandler OrderCreated;
}
```

```
var processor = new OrderProcessor();
```

```
processor.OrderCreated += Processor_OrderCreated;
```

Don't forget to **unsubscribe** when you are done with the **event**!

...

```
processor.OrderCreated -= Processor_OrderCreated;
```



Event **handlers** are **executed**  
**sequentially** in the **order**  
**they were added**



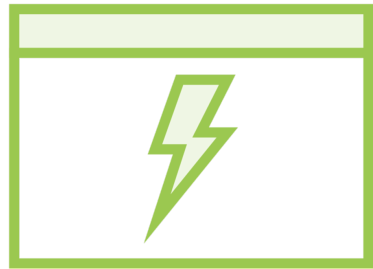
# Multiple Events Are Not Uncommon

```
class OrderProcessor
{
    public event EventHandler OrderCreated;
    public event EventHandler OrderReadyForShipment;
    public event EventHandler OrderCancelled;
    public event EventHandler PaymentProcessed;
}
```

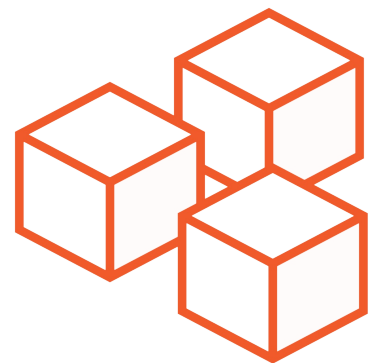




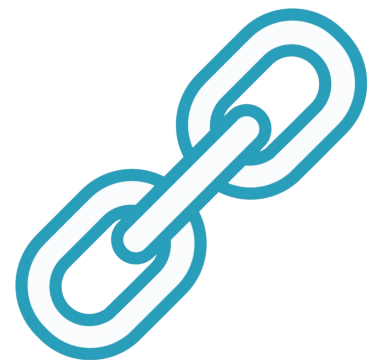
# Important Notes on Events



**The publisher owns the invocation.  
It cannot be invoked by anyone else.**



**Use a delegate that conforms with the .NET guidelines.  
This is EventHandler or EventHandler<T>**



**Subscribers may be attached for a very long time.  
This is especially true in UI applications.**



# EventArgs

```
class OrderCreatedEventArgs : EventArgs
{
    public Order Order { get; set; }
}
```



# EventArgs

```
class OrderCreatedEventArgs : EventArgs
{
    public Order Order { get; set; }
}
```

```
// The event handler
void Processor_OrderCreated(object sender, EventArgs args)
{
    var eventData = args as OrderCreatedEventArgs;
}
```



# Creating an **Event** with **Event Data**

```
class OrderProcessor
{
    public event EventHandler OrderCreated;
}
```



# Creating an **Event** with **Event Data**

```
class OrderProcessor
{
    public event EventHandler OrderCreated;
    public event EventHandler<OrderCreatedEventArgs> OrderCreated;
}
```



# Creating an **Event** with **Event Data**

```
class OrderProcessor
{
    public event EventHandler OrderCreated;
    public event EventHandler<OrderCreatedEventArgs> OrderCreated;
}
```



**Does not have to inherit from EventArgs**



# The parameter support contravariance

A method using EventArgs could therefore be used as a delegate if the event data class inherits from it



**Different parts** of the system  
may **subscribe** to the **event**  
and **handle** it in **different**  
**ways**





**Event handlers** are **delegates**  
and **execute** on the **same**  
**thread** as they were **called on**



# Event with Event Data

```
class OrderProcessor
{
    public event EventHandler<OrderCreatedEventArgs> OrderCreated;

    protected virtual void OnOrderCreated(OrderCreatedEventArgs args)
    {
        OrderCreated?.Invoke(this, args);
    }
}
```



# Warehouse Management System

Warehouse Management System

OrderNumber	IsReadyForShipment	Total
8101e88e-ff6a-4eb7-af07-bdaef0b711e6	<input checked="" type="checkbox"/>	180
0b7c7c69-c46a-4585-a3be-e4f3f0fe96d1	<input checked="" type="checkbox"/>	170

Process Order

# Warehouse Management System

The diagram illustrates two windows that share a reference to the **OrderProcessor**.

**Warehouse Management System**

OrderNumber	IsReadyForShipment	Total
8101e88e-ff6a-4eb7-af07-bdaef0b711e6	<input checked="" type="checkbox"/>	180
0b7c7c69-c46a-4585-a3be-e4f3f0fe96d1	<input checked="" type="checkbox"/>	170

**Process Order**

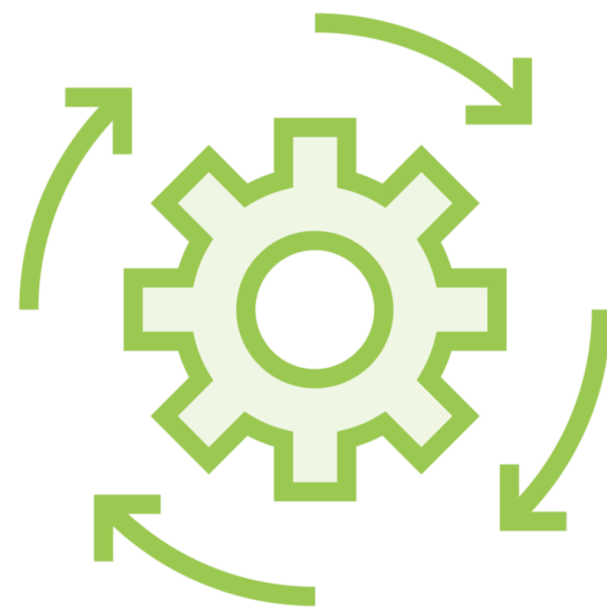
**Order Receipt**

Order is processing, the receipt will be ready soon.

These windows **share a reference** to the **OrderProcessor**

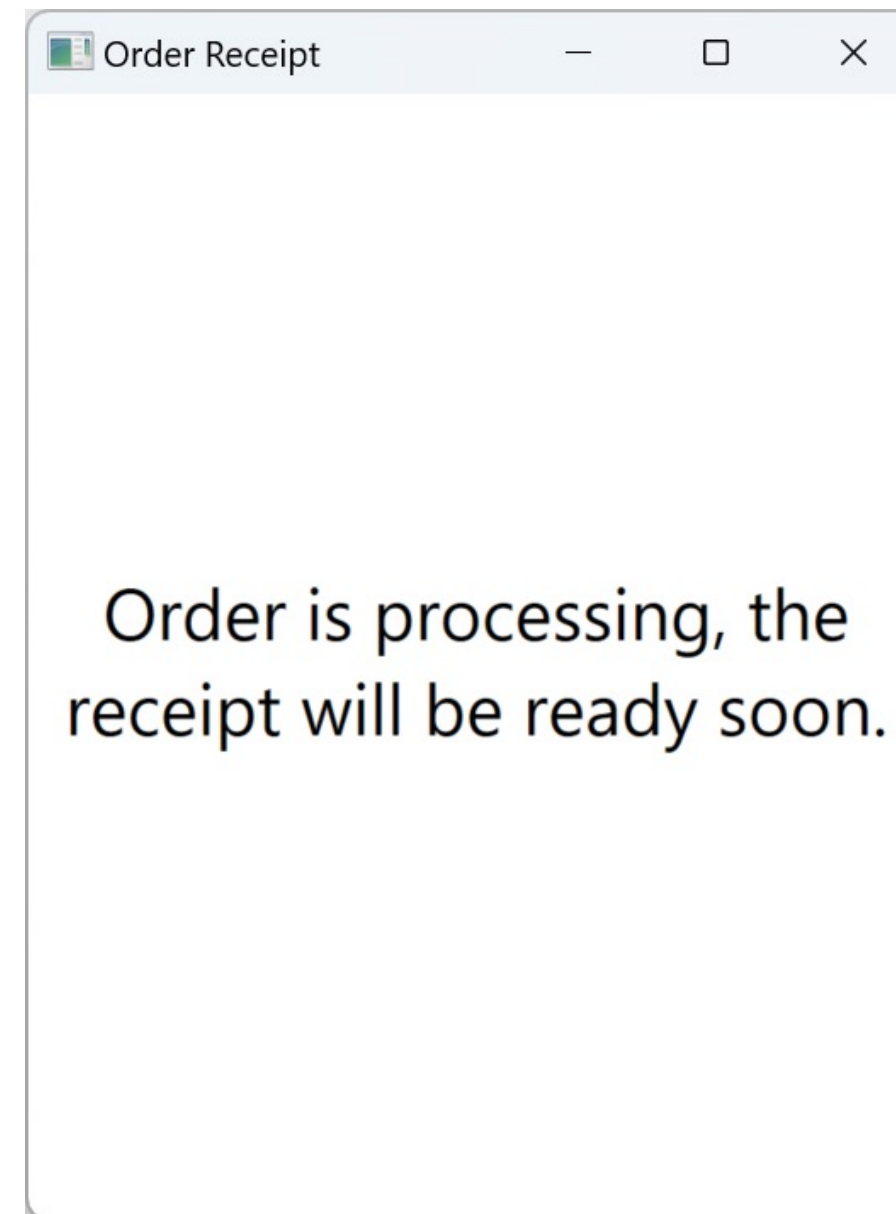


# Warehouse Management System

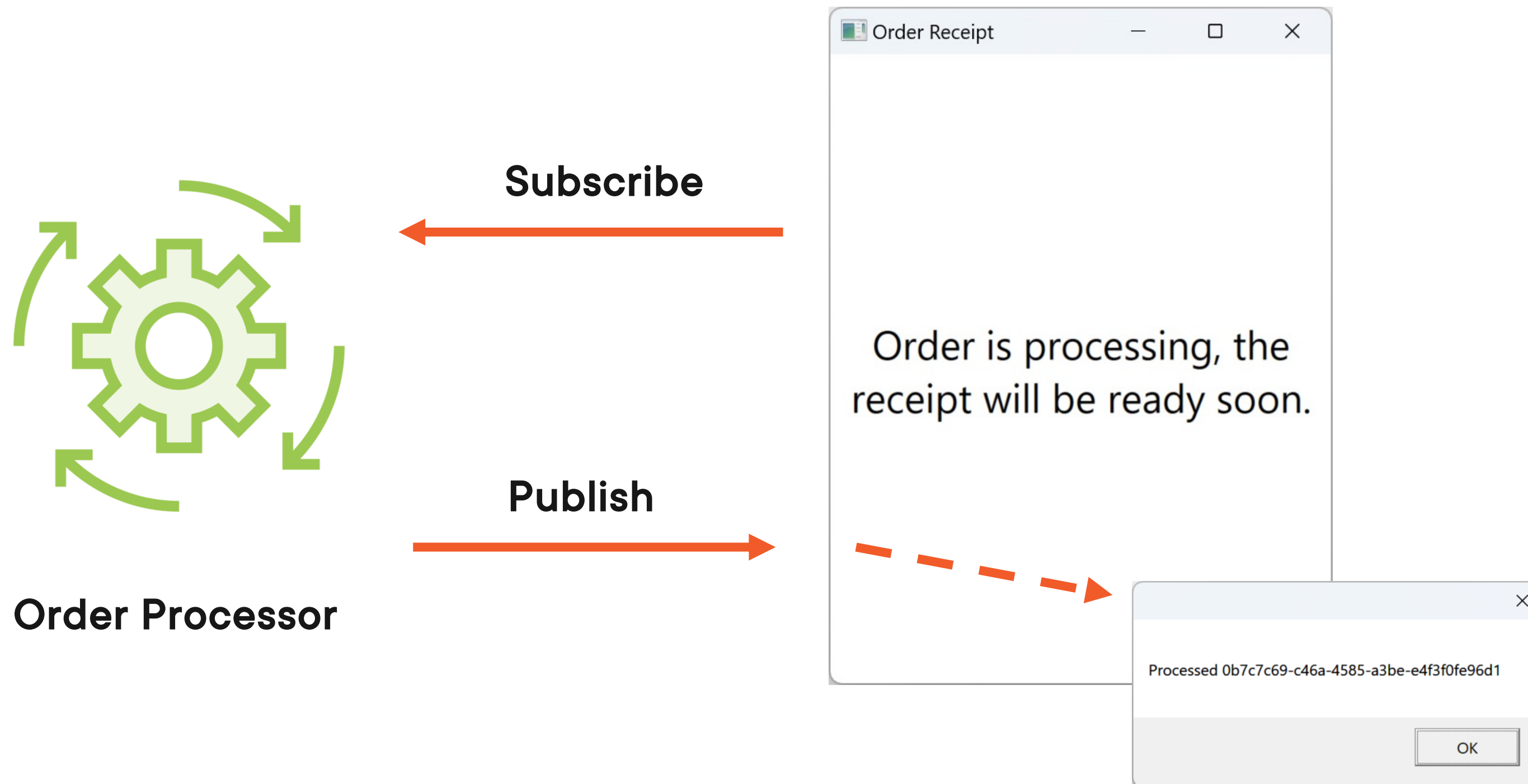


**Order Processor**

**Subscribe**



# Warehouse Management System



**Why was the event handler  
invoked three times?**



We forgot to **unsubscribe!**





**Avoid lambdas for  
event handlers**



Event **handler leaks** can  
**happen** in **any type** of  
**application**



# Event Handler Leaks



**Always unsubscribe from events when the subscriber is no longer relevant**

View closing? Unsubscribe from all events!



**Where** you **unsubscribe** will  
**depend** on the **type of**  
**application** and how the  
subscriber works



# Alternative Approach

```
var processor = new OrderProcessor();  
var logger = new Logger();  
  
processor.OrderCreated += Processor_OrderCreated;  
  
...  
  
processor.OrderCreated -= Processor_OrderCreated;  
  
void Processor_OrderCreated(object sender, OrderCreatedEventArgs args)  
{  
    logger.Log(args.Order);  
}
```



# Understanding **Events**



Understanding of delegates is required



The publisher owns the invocation it cannot even be invoked by a sub-class



Use the **event** keyword with a delegate of type **EventHandler** or **EventHandler<T>**



# The **Event** Keyword

```
class OrderProcessor
{
    public event EventHandler OrderCreated;
}
```



# The **Event** Keyword

```
class OrderProcessor
{
    public event EventHandler OrderCreated;
}
```

External **access** is **limited** to **adding** and **removing delegates** from the invocation list





# Raising the **Event**

```
class OrderProcessor
{
    public event EventHandler OrderCreated;

    protected virtual void OnOrderCreated()
    {
        OrderCreated?.Invoke();
    }
}
```



# Raising the **Event**

```
class OrderProcessor
{
    public event EventHandler OrderCreated;

    protected virtual void OnOrderCreated()
    {
        OrderCreated?.Invoke();
    }
}
```



**Always call the base class to  
raise the event!**



# Benefit of Inheriting from **EventArgs**

```
public void Log(object sender, EventArgs args) {}
```

```
public void Log(object sender, OrderCreatedEventArgs args) {}
```



# Benefit of Inheriting from **EventArgs**

```
public void Log(object sender, EventArgs args) {}
```

```
public void Log(object sender, OrderCreatedEventArgs args) {}
```



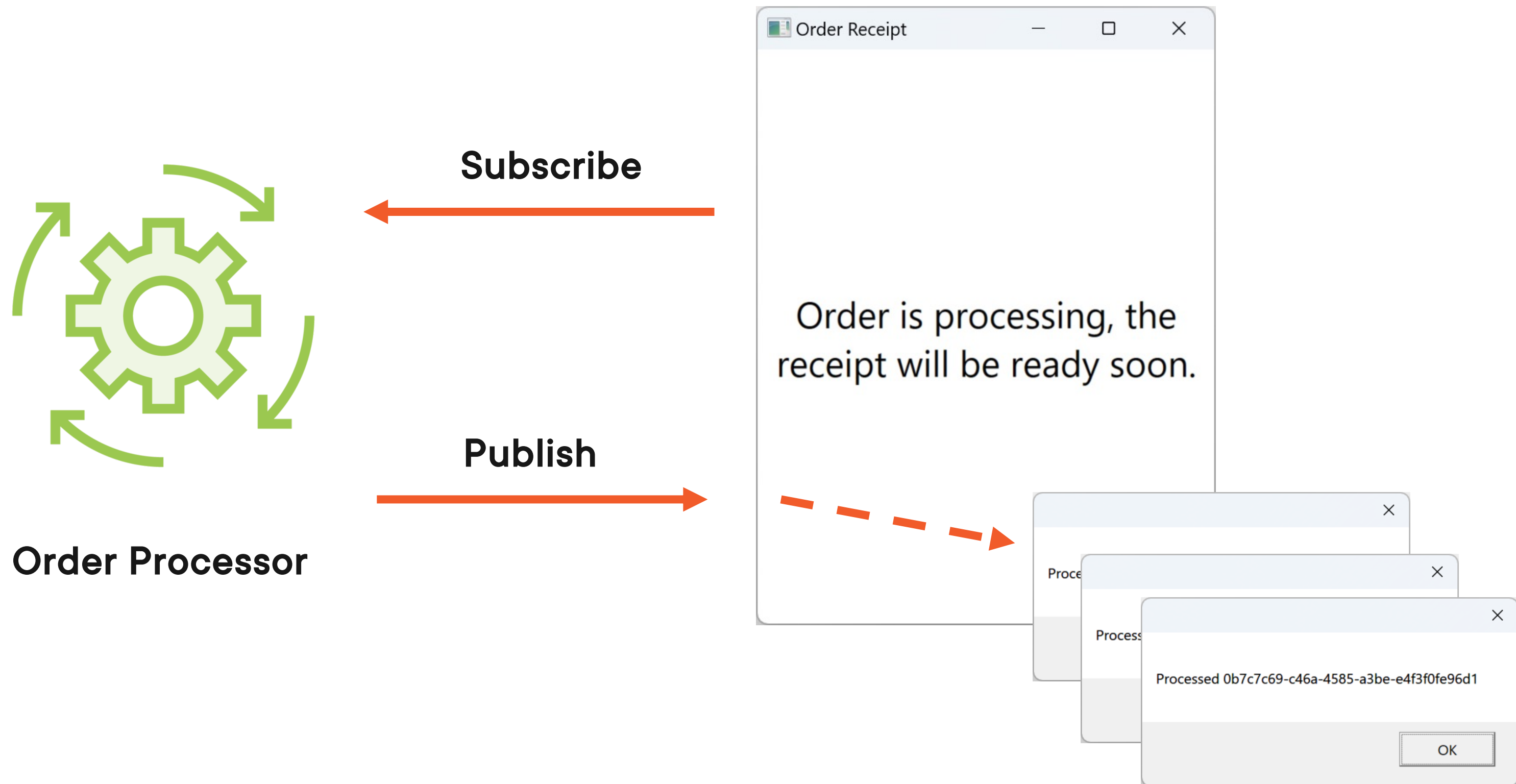
If this inherits from **EventArgs** it could be **used with any** event that uses **EventHandler** or **EventHandler<T>**



**Using events in user  
interfaces are very common**



# Warehouse Management System



# Next: **Overloading and Extension Methods**

---

