upGrad ENTERPRISE

Events & Publisher-Subscriber

By
Narasimha Rao T
Microsoft.Net FSD Trainer
Professional Development Trainer
tnrao.trainer@gmail.com

Events and the Publisher-Subscriber Model in C#

1. Publisher-Subscriber (Pub-Sub) Model

- A design pattern where a **publisher** sends messages or events and one or more **subscribers** receive them.
- Promotes **loose coupling** between components.

Key Roles

- **Publisher**: Class that defines and raises an event.
- **Subscriber**: Class that responds to the event.

2. `event` Keyword in C#

- Declares an event in the publisher.
- Prevents external classes from invoking the event directly.

```
public class Publisher
{
    public event EventHandler MyEvent;

    public void RaiseEvent()
    {
        MyEvent?.Invoke(this, EventArgs.Empty);
    }
}
```

3. `EventHandler` and `EventHandler<T>`

`EventHandler`

A predefined delegate with this signature:

```
void Handler(object sender, EventArgs e);
```

`EventHandler<T>`

Used when you want to pass custom data.

```
public class CustomEventArgs : EventArgs
{
    public string Message { get; set; }
}

public event EventHandler<CustomEventArgs> OnDataProcessed;
```

4. Creating Custom `EventArgs` Classes

```
public class ThresholdReachedEventArgs : EventArgs
{
    public int Threshold { get; set; }
    public DateTime TimeReached { get; set; }
}
```

5. Subscribing & Unsubscribing to Events

Subscribing

```
publisher.MyEvent += HandlerMethod;
```

Unsubscribing

```
publisher.MyEvent -= HandlerMethod;
```

Example Handler

```
private void HandlerMethod(object sender, EventArgs e)
{
    Console.WriteLine("Event triggered!");
}
```

6. Real-World Event Simulation Examples

Button Click Simulation

```
public class Button
{
    public event EventHandler Click;

    public void Press()
    {
        Console.WriteLine("Button pressed!");
        Click?.Invoke(this, EventArgs.Empty);
    }
}
```

```
Button button = new Button();
button.Click += (s, e) => Console.WriteLine("Button click handled.");
button.Press();
```

7. Best Practices

- Always check if event is null before invoking: `?.Invoke(...)`
- Use custom `EventArgs` for meaningful event data.
- Unsubscribe from events to prevent memory leaks, especially in long-lived applications like WPF or WinForms.

Real-Time Use Cases

1. **UI Button Click** – Event triggers when a button is pressed.
2. **Stock Price Monitor** – Alert when price crosses a threshold.
3. **File System Watcher** – Trigger event when a file is changed.
4. **Gaming Engine** – Event when player scores or takes damage.
5. **E-commerce** – Notify when order is shipped.

Quiz

1. What is the main advantage of the Publisher-Subscriber model?
2. What does the `event` keyword prevent in C#?
3. What is the signature of the `EventHandler` delegate?
4. How do you pass custom data through an event?
5. What happens if you try to raise an event when no handlers are attached?

upGrad ENTERPRISE

Q & A

Narasimha Rao T
tnrao.trainer@gmail.com