Real-World Example: Online Shopping System with Events & Delegates

Complete Working Project

This example demonstrates a realistic online shopping system using events and delegates for:

- Order notifications
- Payment processing
- Inventory management
- Email notifications
- SMS alerts

Complete Code

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace OnlineShoppingSystem
{
    #region Event Arguments Classes

    // Event data for order events
    class OrderEventArgs : EventArgs
    {
        public int OrderId { get; set; }
        public string CustomerName { get; set; }
        public decimal TotalAmount { get; set; }
        public DateTime OrderDate { get; set; }
        public List<OrderItem> Items { get; set; }

        public OrderEventArgs(int orderId, string customerName, decimal
totalAmount, List<OrderItem> items)
        {
            OrderId = orderId;
            CustomerName = customerName;
            TotalAmount = totalAmount;
            OrderDate = DateTime.Now;
            Items = items;
        }
    }

    // Event data for payment events
    class PaymentEventArgs : EventArgs
    {
        public int OrderId { get; set; }
        public decimal Amount { get; set; }
        public string PaymentMethod { get; set; }
        public bool IsSuccessful { get; set; }
        public string TransactionId { get; set; }

        public PaymentEventArgs(int orderId, decimal amount, string
paymentMethod, bool isSuccessful)
        {
            OrderId = orderId;
            Amount = amount;
            PaymentMethod = paymentMethod;
            IsSuccessful = isSuccessful;
            TransactionId = Guid.NewGuid().ToString().Substring(0, 8);
        }
    }

    // Event data for shipping events
    class ShippingEventArgs : EventArgs
    {
        public int OrderId { get; set; }
        public string Address { get; set; }
        public string TrackingNumber { get; set; }
        public DateTime EstimatedDelivery { get; set; }

        public ShippingEventArgs(int orderId, string address)
        {
            OrderId = orderId;
            Address = address;
            TrackingNumber = $"TRACK-{Guid.NewGuid().ToString().Substring(0,
8)}";
            EstimatedDelivery = DateTime.Now.AddDays(3);
        }
    }

    #endregion

    #region Supporting Classes

    class OrderItem
    {
        public int ProductId { get; set; }
        public string ProductName { get; set; }
        public int Quantity { get; set; }
        public decimal Price { get; set; }
        public decimal Subtotal => Quantity * Price;

        public override string ToString()
        {
            return $"  {ProductName} x{Quantity} @ ${Price} = ${Subtotal}";
        }
    }

    class Product
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public decimal Price { get; set; }
        public int Stock { get; set; }

        public override string ToString()
        {
            return $"{Name} (${Price}) - Stock: {Stock}";
        }
    }

    #endregion

    #region Publisher Class - Order System

    class OrderSystem
    {
        // Events - using EventHandler<T>
        public event EventHandler<OrderEventArgs> OrderPlaced;
        public event EventHandler<OrderEventArgs> OrderCancelled;
        public event EventHandler<PaymentEventArgs> PaymentProcessed;
        public event EventHandler<ShippingEventArgs> OrderShipped;

        private int nextOrderId = 1000;
        private List<Product> products;

        public OrderSystem()
        {
            InitializeProducts();
        }
        private void InitializeProducts()
        {
            products = new List<Product>
            {
                new Product { Id = 1, Name = "Laptop", Price = 999.99m, Stock
= 10 },
                new Product { Id = 2, Name = "Mouse", Price = 29.99m, Stock =
50 },
                new Product { Id = 3, Name = "Keyboard", Price = 79.99m, Stock
= 30 },
                new Product { Id = 4, Name = "Monitor", Price = 299.99m, Stock
= 15 },
                new Product { Id = 5, Name = "Headphones", Price = 149.99m,
Stock = 25 }
            };
        }

        public void DisplayProducts()
        {
            Console.WriteLine("\n=== Available Products ===");
            foreach (var product in products)
            {
                Console.WriteLine($"{product.Id}. {product}");
            }
            Console.WriteLine();
        }

        public Product GetProduct(int id)
        {
            return products.FirstOrDefault(p => p.Id == id);
        }

        // Place new order
        public void PlaceOrder(string customerName, List<OrderItem> items)
        {
            int orderId = nextOrderId++;
            decimal totalAmount = items.Sum(i => i.Subtotal);

            Console.WriteLine($"\n{'=',50}");
            Console.WriteLine($"  PROCESSING ORDER #{orderId}");
            Console.WriteLine($"{'=',50}\n");

            // Check stock availability
            bool stockAvailable = CheckStock(items);
            if (!stockAvailable)
            {
                Console.WriteLine("❌ Order failed: Insufficient stock!\n");
                return;
            }

            // Reduce stock
            UpdateStock(items);

            // Raise OrderPlaced event
            OnOrderPlaced(new OrderEventArgs(orderId, customerName,
totalAmount, items));
        }

        private bool CheckStock(List<OrderItem> items)
        {
            foreach (var item in items)
            {
                var product = GetProduct(item.ProductId);
                if (product == null || product.Stock < item.Quantity)
                {
                    return false;
                }
            }
            return true;
        }

        private void UpdateStock(List<OrderItem> items)
        {
            foreach (var item in items)
            {
                var product = GetProduct(item.ProductId);
                if (product != null)
                {
                    product.Stock -= item.Quantity;
                }
            }
        }

        // Process payment
        public void ProcessPayment(int orderId, decimal amount, string
paymentMethod)
        {
            // Simulate payment processing
            bool isSuccessful = new Random().Next(0, 10) > 1; // 90% success
rate

            OnPaymentProcessed(new PaymentEventArgs(orderId, amount,
paymentMethod, isSuccessful));
        }

        // Ship order
        public void ShipOrder(int orderId, string address)
        {
            OnOrderShipped(new ShippingEventArgs(orderId, address));
        }

        // Cancel order
        public void CancelOrder(int orderId, string customerName,
List<OrderItem> items, decimal totalAmount)
        {
            // Restore stock
            foreach (var item in items)
            {
                var product = GetProduct(item.ProductId);
                if (product != null)
                {
                    product.Stock += item.Quantity;
                }
            }

            OnOrderCancelled(new OrderEventArgs(orderId, customerName,
totalAmount, items));
        }

        // Protected methods to raise events
        protected virtual void OnOrderPlaced(OrderEventArgs e)
        {
            OrderPlaced?.Invoke(this, e);
        }

        protected virtual void OnOrderCancelled(OrderEventArgs e)
        {
            OrderCancelled?.Invoke(this, e);
        }

        protected virtual void OnPaymentProcessed(PaymentEventArgs e)
        {
            PaymentProcessed?.Invoke(this, e);
        }

        protected virtual void OnOrderShipped(ShippingEventArgs e)
        {
            OrderShipped?.Invoke(this, e);
        }
    }

    #endregion

    #region Subscriber Classes

    // Email notification service
    class EmailService
    {
        public string ServiceName { get; set; } = "Email Service";

        public void SendOrderConfirmation(object sender, OrderEventArgs e)
        {
            Console.WriteLine($"\n📧 [{ServiceName}] Sending email to
customer...");
            Console.WriteLine($"   To: {e.CustomerName}");
            Console.WriteLine($"   Subject: Order #{e.OrderId} Confirmation");
            Console.WriteLine($"   Order Details:");
            foreach (var item in e.Items)
            {
                Console.WriteLine($"   {item}");
            }
            Console.WriteLine($"   Total: ${e.TotalAmount:F2}");
            Console.WriteLine($"   ✅ Email sent successfully!");
        }

        public void SendPaymentReceipt(object sender, PaymentEventArgs e)
        {
            if (e.IsSuccessful)
            {
                Console.WriteLine($"\n📧 [{ServiceName}] Payment receipt
sent");
                Console.WriteLine($"   Transaction ID: {e.TransactionId}");
                Console.WriteLine($"   Amount: ${e.Amount:F2}");
                Console.WriteLine($"   Method: {e.PaymentMethod}");
            }
            else
            {
                Console.WriteLine($"\n📧 [{ServiceName}] Payment failed
notification sent");
            }
        }

        public void SendShippingNotification(object sender, ShippingEventArgs
e)
```

```
        {
            Console.WriteLine($"\n📧 [{ServiceName}] Shipping notification
sent");
            Console.WriteLine($"   Order #{e.OrderId} has been shipped!");
            Console.WriteLine($"   Tracking: {e.TrackingNumber}");
            Console.WriteLine($"   Estimated Delivery:
{e.EstimatedDelivery:yyyy-MM-dd}");
        }

        public void SendCancellationEmail(object sender, OrderEventArgs e)
        {
            Console.WriteLine($"\n📧 [{ServiceName}] Cancellation email
sent");
            Console.WriteLine($"   Order #{e.OrderId} has been cancelled");
            Console.WriteLine($"   Refund amount: ${e.TotalAmount:F2}");
        }
    }

    // SMS notification service
    class SmsService
    {
        public string ServiceName { get; set; } = "SMS Service";

        public void SendOrderSms(object sender, OrderEventArgs e)
        {
            Console.WriteLine($"\n📱 [{ServiceName}] SMS sent to customer");
            Console.WriteLine($"   Message: Your order #{e.OrderId} has been
placed successfully!");
            Console.WriteLine($"   Total: ${e.TotalAmount:F2}");
        }

        public void SendPaymentSms(object sender, PaymentEventArgs e)
        {
            if (e.IsSuccessful)
            {
                Console.WriteLine($"\n📱 [{ServiceName}] Payment confirmed via
SMS");
                Console.WriteLine($"   Message: Payment of ${e.Amount:F2}
received. Thank you!");
            }
        }

        public void SendShippingSms(object sender, ShippingEventArgs e)
        {
            Console.WriteLine($"\n📱 [{ServiceName}] Shipping SMS sent");
            Console.WriteLine($"   Message: Order #{e.OrderId} shipped! Track:
```

```
{e.TrackingNumber}");
        }
    }

    // Inventory management service
    class InventoryService
    {
        public string ServiceName { get; set; } = "Inventory Service";
        private int lowStockThreshold = 10;

        public void UpdateInventory(object sender, OrderEventArgs e)
        {
            Console.WriteLine($"\n📦 [{ServiceName}] Inventory updated");

            OrderSystem orderSystem = sender as OrderSystem;
            foreach (var item in e.Items)
            {
                var product = orderSystem?.GetProduct(item.ProductId);
                if (product != null)
                {
                    Console.WriteLine($"   {product.Name}: {product.Stock}
units remaining");

                    if (product.Stock < lowStockThreshold)
                    {
                        Console.WriteLine($"   ⚠️  LOW STOCK ALERT for
{product.Name}!");
                    }
                }
            }
        }

        public void RestoreInventory(object sender, OrderEventArgs e)
        {
            Console.WriteLine($"\n📦 [{ServiceName}] Inventory restored");
            Console.WriteLine($"   Items returned to stock for cancelled order
#{e.OrderId}");
        }
    }

    // Accounting/Finance service
    class AccountingService
    {
        public string ServiceName { get; set; } = "Accounting Service";
        private decimal totalRevenue = 0;
        public void RecordTransaction(object sender, PaymentEventArgs e)
        {
            if (e.IsSuccessful)
            {
                totalRevenue += e.Amount;
                Console.WriteLine($"\n💰 [{ServiceName}] Transaction
recorded");
                Console.WriteLine($"   Order ID: {e.OrderId}");
                Console.WriteLine($"   Amount: ${e.Amount:F2}");
                Console.WriteLine($"   Payment Method: {e.PaymentMethod}");
                Console.WriteLine($"   Total Revenue: ${totalRevenue:F2}");
            }
            else
            {
                Console.WriteLine($"\n💰 [{ServiceName}] Payment failed - No
transaction recorded");
            }
        }

        public void ProcessRefund(object sender, OrderEventArgs e)
        {
            totalRevenue -= e.TotalAmount;
            Console.WriteLine($"\n💰 [{ServiceName}] Refund processed");
            Console.WriteLine($"   Order #{e.OrderId}");
            Console.WriteLine($"   Refund Amount: ${e.TotalAmount:F2}");
            Console.WriteLine($"   Total Revenue: ${totalRevenue:F2}");
        }
    }

    // Shipping/Logistics service
    class ShippingService
    {
        public string ServiceName { get; set; } = "Shipping Service";

        public void PrepareShipment(object sender, ShippingEventArgs e)
        {
            Console.WriteLine($"\n🚚 [{ServiceName}] Preparing shipment");
            Console.WriteLine($"   Order #{e.OrderId}");
            Console.WriteLine($"   Destination: {e.Address}");
            Console.WriteLine($"   Tracking Number: {e.TrackingNumber}");
            Console.WriteLine($"   Estimated Delivery:
{e.EstimatedDelivery:yyyy-MM-dd}");
            Console.WriteLine($"   ✅ Package ready for pickup!");
        }
    }
```

```csharp
    // Analytics/Reporting service
    class AnalyticsService
    {
        public string ServiceName { get; set; } = "Analytics Service";
        private int totalOrders = 0;
        private int successfulPayments = 0;
        private int failedPayments = 0;
        private int cancelledOrders = 0;

        public void TrackOrder(object sender, OrderEventArgs e)
        {
            totalOrders++;
            Console.WriteLine($"\n📊 [{ServiceName}] Order tracked");
            Console.WriteLine($"   Total Orders Today: {totalOrders}");
        }

        public void TrackPayment(object sender, PaymentEventArgs e)
        {
            if (e.IsSuccessful)
                successfulPayments++;
            else
                failedPayments++;

            Console.WriteLine($"\n📊 [{ServiceName}] Payment statistics
updated");
            Console.WriteLine($"   Successful: {successfulPayments}");
            Console.WriteLine($"   Failed: {failedPayments}");
            Console.WriteLine($"   Success Rate: {GetSuccessRate():F1}%");
        }

        public void TrackCancellation(object sender, OrderEventArgs e)
        {
            cancelledOrders++;
            Console.WriteLine($"\n📊 [{ServiceName}] Cancellation tracked");
            Console.WriteLine($"   Cancelled Orders: {cancelledOrders}");
        }

        private double GetSuccessRate()
        {
            int total = successfulPayments + failedPayments;
            return total > 0 ? (successfulPayments * 100.0 / total) : 0;
        }
    }

    #endregion
    #region Main Program

    class Program
    {
        static void Main(string[] args)
        {

Console.WriteLine("╔═════════════════════════════════════════════════════╗");
            Console.WriteLine("║     ONLINE SHOPPING SYSTEM - EVENTS DEMO
║");

Console.WriteLine("╚═════════════════════════════════════════════════════╝\n");

            // Create the order system (Publisher)
            OrderSystem orderSystem = new OrderSystem();

            // Create services (Subscribers)
            EmailService emailService = new EmailService();
            SmsService smsService = new SmsService();
            InventoryService inventoryService = new InventoryService();
            AccountingService accountingService = new AccountingService();
            ShippingService shippingService = new ShippingService();
            AnalyticsService analyticsService = new AnalyticsService();

            // Subscribe to events
            Console.WriteLine("🔗 Setting up event subscriptions...\n");

            // OrderPlaced event - Multiple subscribers
            orderSystem.OrderPlaced += emailService.SendOrderConfirmation;
            orderSystem.OrderPlaced += smsService.SendOrderSms;
            orderSystem.OrderPlaced += inventoryService.UpdateInventory;
            orderSystem.OrderPlaced += analyticsService.TrackOrder;

            // PaymentProcessed event
            orderSystem.PaymentProcessed += emailService.SendPaymentReceipt;
            orderSystem.PaymentProcessed += smsService.SendPaymentSms;
            orderSystem.PaymentProcessed +=
accountingService.RecordTransaction;
            orderSystem.PaymentProcessed += analyticsService.TrackPayment;

            // OrderShipped event
            orderSystem.OrderShipped += emailService.SendShippingNotification;
            orderSystem.OrderShipped += smsService.SendShippingSms;
            orderSystem.OrderShipped += shippingService.PrepareShipment;

            // OrderCancelled event
            orderSystem.OrderCancelled += emailService.SendCancellationEmail;
            orderSystem.OrderCancelled += inventoryService.RestoreInventory;
            orderSystem.OrderCancelled += accountingService.ProcessRefund;
            orderSystem.OrderCancelled += analyticsService.TrackCancellation;

            Console.WriteLine("✅ All services subscribed successfully!\n");

            // Display available products
            orderSystem.DisplayProducts();

            // Scenario 1: Successful Order
            Console.WriteLine("\n" + new string('=', 60));
            Console.WriteLine("SCENARIO 1: Successful Order");
            Console.WriteLine(new string('=', 60));

            List<OrderItem> order1Items = new List<OrderItem>
            {
                new OrderItem { ProductId = 1, ProductName = "Laptop",
Quantity = 1, Price = 999.99m },
                new OrderItem { ProductId = 2, ProductName = "Mouse", Quantity
= 2, Price = 29.99m }
            };

            orderSystem.PlaceOrder("Ahmed Ali", order1Items);
            orderSystem.ProcessPayment(1000, 1059.97m, "Credit Card");
            orderSystem.ShipOrder(1000, "123 Main St, Cairo, Egypt");

            // Scenario 2: Failed Payment
            Console.WriteLine("\n\n" + new string('=', 60));
            Console.WriteLine("SCENARIO 2: Order with Failed Payment");
            Console.WriteLine(new string('=', 60));

            List<OrderItem> order2Items = new List<OrderItem>
            {
                new OrderItem { ProductId = 3, ProductName = "Keyboard",
Quantity = 1, Price = 79.99m }
            };

            orderSystem.PlaceOrder("Sara Mohamed", order2Items);
            orderSystem.ProcessPayment(1001, 79.99m, "PayPal");

            // Scenario 3: Order Cancellation
            Console.WriteLine("\n\n" + new string('=', 60));
            Console.WriteLine("SCENARIO 3: Order Cancellation");
            Console.WriteLine(new string('=', 60));
```

```csharp
            List<OrderItem> order3Items = new List<OrderItem>
            {
                new OrderItem { ProductId = 4, ProductName = "Monitor",
Quantity = 1, Price = 299.99m }
            };

            orderSystem.PlaceOrder("Omar Hassan", order3Items);
            orderSystem.ProcessPayment(1002, 299.99m, "Debit Card");

            // Customer cancels order
            Console.WriteLine("\n📞 Customer requests cancellation...");
            orderSystem.CancelOrder(1002, "Omar Hassan", order3Items,
299.99m);

            // Scenario 4: Multiple Items Order
            Console.WriteLine("\n\n" + new string('=', 60));
            Console.WriteLine("SCENARIO 4: Large Order (Multiple Items)");
            Console.WriteLine(new string('=', 60));

            List<OrderItem> order4Items = new List<OrderItem>
            {
                new OrderItem { ProductId = 1, ProductName = "Laptop",
Quantity = 2, Price = 999.99m },
                new OrderItem { ProductId = 3, ProductName = "Keyboard",
Quantity = 2, Price = 79.99m },
                new OrderItem { ProductId = 5, ProductName = "Headphones",
Quantity = 1, Price = 149.99m }
            };

            orderSystem.PlaceOrder("Fatma Ibrahim", order4Items);
            orderSystem.ProcessPayment(1003, 2309.95m, "Credit Card");
            orderSystem.ShipOrder(1003, "456 Nile St, Alexandria, Egypt");

            // Display final inventory
            Console.WriteLine("\n\n" + new string('=', 60));
            Console.WriteLine("FINAL INVENTORY STATUS");
            Console.WriteLine(new string('=', 60));
            orderSystem.DisplayProducts();


Console.WriteLine("\n\n╔════════════════════════════════════════════════════╗"
);
            Console.WriteLine("║              DEMO COMPLETED!
║");

Console.WriteLine("╚════════════════════════════════════════════════════╝");
```

```
            Console.WriteLine("\nPress any key to exit...");
            Console.ReadKey();
        }
    }

    #endregion
}
```

How to Run

1. Create a new C# Console Application
2. Copy the entire code above
3. Run the program
4. Observe how events trigger multiple services automatically!

What This Example Demonstrates

1. Events in Action

```
When order is placed:
┌─────────────────────────────────────┐
│ OrderSystem.PlaceOrder()            │
│    ↓                                │
│ Raises OrderPlaced event            │
│    ↓                                │
│ ALL subscribed services notified:   │
│  • Email Service                    │
│  • SMS Service                      │
│  • Inventory Service                │
│  • Analytics Service                │
└─────────────────────────────────────┘
```

2. Loose Coupling

- OrderSystem doesn't know about services
- Services don't know about each other
- Easy to add/remove services
- No code changes in OrderSystem needed

3. Real-World Scenarios

✅ **Successful Order Flow:**

- Order placed → All services notified
- Payment processed → Receipt sent
- Order shipped → Tracking info sent

✅ **Failed Payment:**

- Payment fails → Only relevant services notified
- No shipping triggers

✅ **Order Cancellation:**

- Inventory restored
- Refund processed
- Customer notified

4. Multiple Subscribers

Each event can have multiple handlers:

- `OrderPlaced` → 4 handlers (Email, SMS, Inventory, Analytics)
- `PaymentProcessed` → 4 handlers
- Easy to add more services!

Key Takeaways

✓ **Benefits Demonstrated**

1. Extensibility

- Add new service? Just subscribe to events!
- No changes to OrderSystem needed

2. Maintainability

- Each service is independent
- Easy to test individually
- Clear separation of concerns

3. Flexibility

- Enable/disable services by subscribing/unsubscribing
- Services can be added at runtime

4. Real-World Patterns

- Email notifications
- SMS alerts
- Inventory tracking
- Analytics/reporting
- Payment processing
- Shipping logistics

Expected Output Sample

```
╔════════════════════════════════════════════════════════╗
║     ONLINE SHOPPING SYSTEM - EVENTS DEMO               ║
╚════════════════════════════════════════════════════════╝

📎 Setting up event subscriptions...

✅ All services subscribed successfully!

=== Available Products ===
1. Laptop ($999.99) - Stock: 10
2. Mouse ($29.99) - Stock: 50
3. Keyboard ($79.99) - Stock: 30
4. Monitor ($299.99) - Stock: 15
5. Headphones ($149.99) - Stock: 25


════════════════════════════════════════════════════════
SCENARIO 1: Successful Order
════════════════════════════════════════════════════════

══════════════════════════════════════════════════
  PROCESSING ORDER #1000
══════════════════════════════════════════════════
```

```
📧 [Email Service] Sending email to customer...
   To: Ahmed Ali
   Subject: Order #1000 Confirmation
   Order Details:
   Laptop x1 @ $999.99 = $999.99
   Mouse x2 @ $29.99 = $59.98
   Total: $1059.97
   ✅ Email sent successfully!

📱 [SMS Service] SMS sent to customer
   Message: Your order #1000 has been placed successfully!
   Total: $1059.97

📦 [Inventory Service] Inventory updated
   Laptop: 9 units remaining
   Mouse: 48 units remaining

📊 [Analytics Service] Order tracked
   Total Orders Today: 1

📧 [Email Service] Payment receipt sent
   Transaction ID: a3f5e9c1
   Amount: $1059.97
   Method: Credit Card

📱 [SMS Service] Payment confirmed via SMS
   Message: Payment of $1059.97 received. Thank you!

💰 [Accounting Service] Transaction recorded
   Order ID: 1000
   Amount: $1059.97
   Payment Method: Credit Card
   Total Revenue: $1059.97

📊 [Analytics Service] Payment statistics updated
   Successful: 1
   Failed: 0
   Success Rate: 100.0%

📧 [Email Service] Shipping notification sent
   Order #1000 has been shipped!
   Tracking: TRACK-b7d2c4e8
   Estimated Delivery: 2026-01-15

📱 [SMS Service] Shipping SMS sent
   Message: Order #1000 shipped! Track: TRACK-b7d2c4e8
```

```
🚚 [Shipping Service] Preparing shipment
   Order #1000
   Destination: 123 Main St, Cairo, Egypt
   Tracking Number: TRACK-b7d2c4e8
   Estimated Delivery: 2026-01-15
   ✅ Package ready for pickup!

... (more scenarios)
```

This is a complete, working example that demonstrates events and delegates in a realistic scenario! 🎉