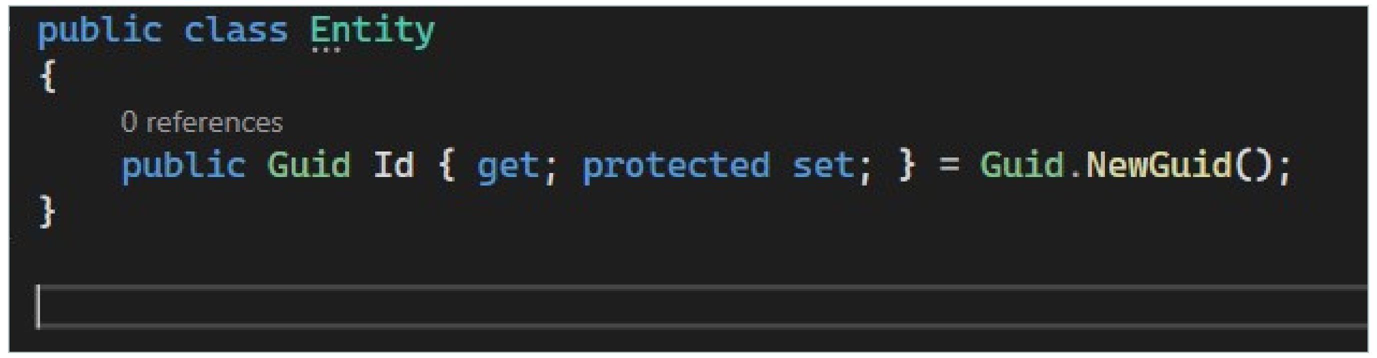
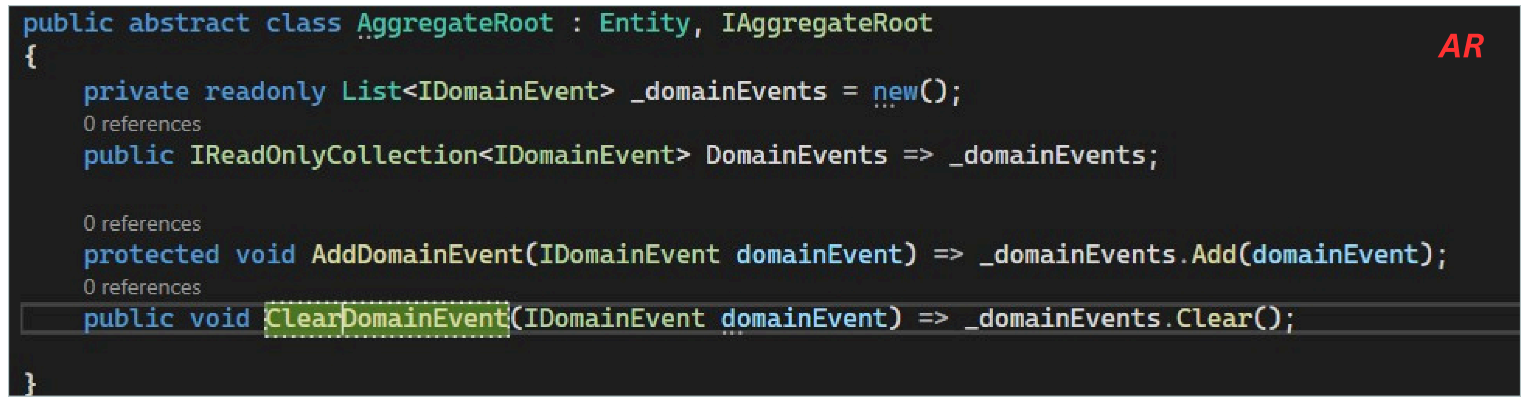
Firuza Polad

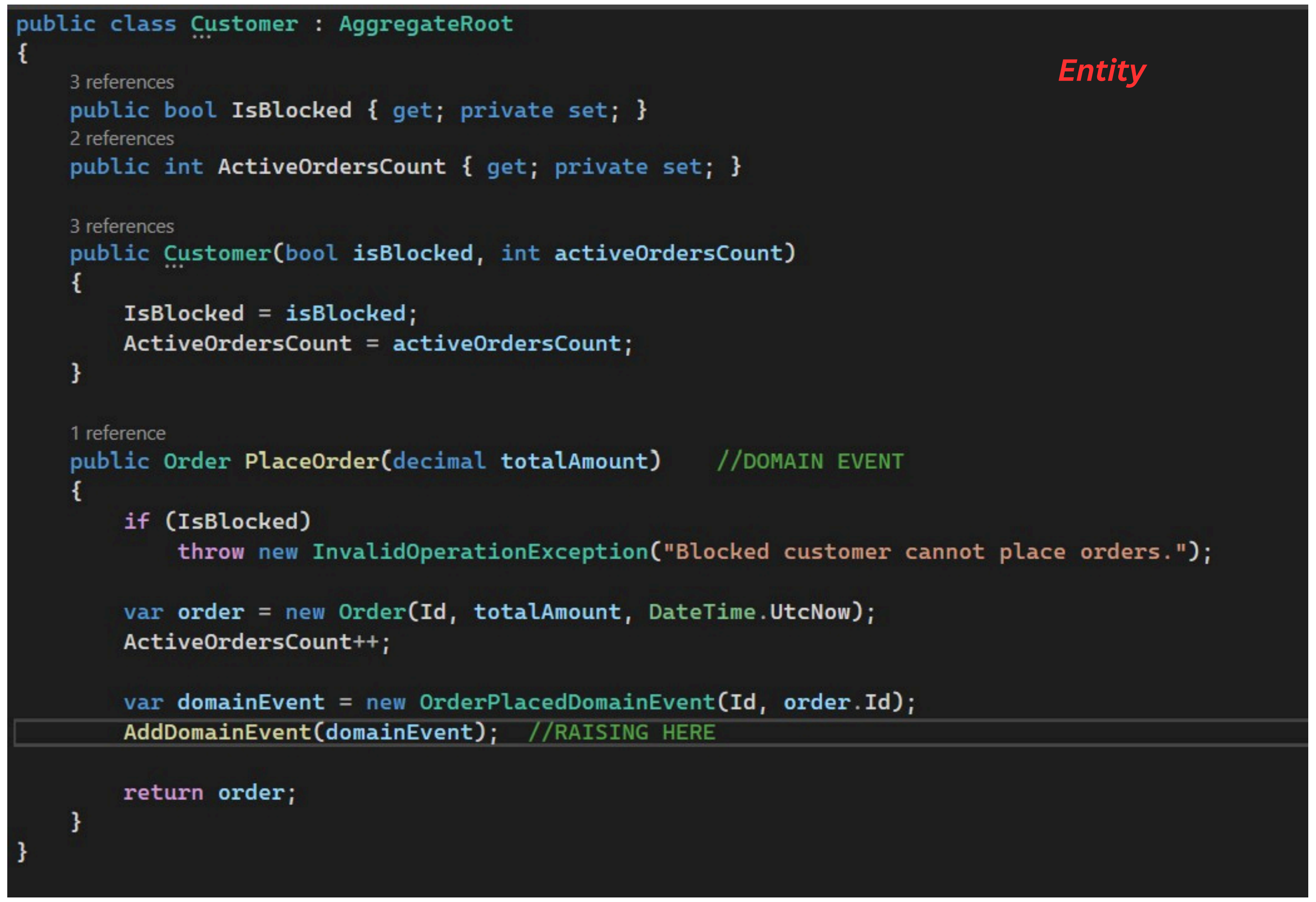


```csharp
public class Entity
{
    0 references
    public Guid Id { get; protected set; } = Guid.NewGuid();
}
```



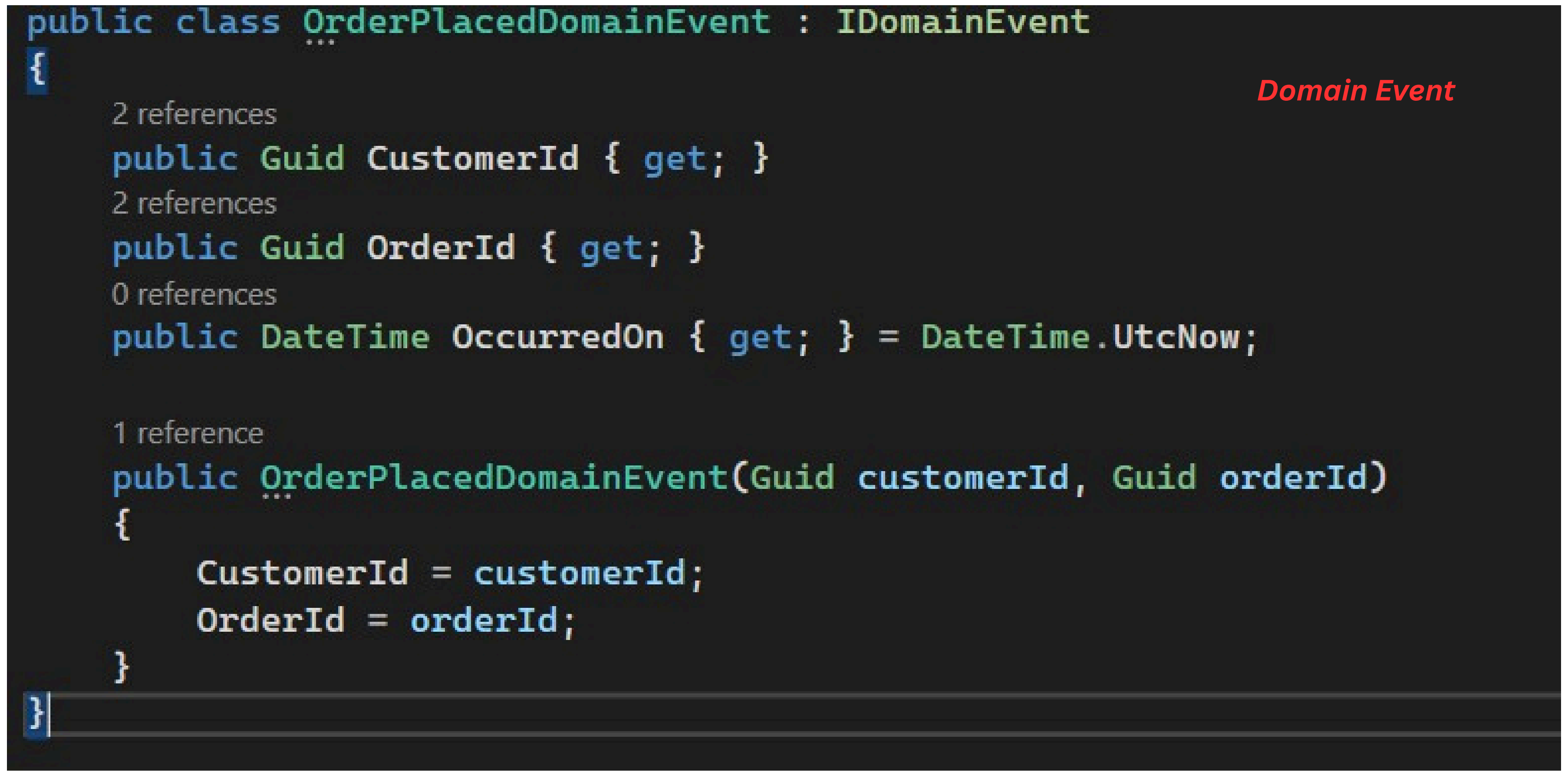
AR

```csharp
public abstract class AggregateRoot : Entity, IAggregateRoot
{
    private readonly List<IDomainEvent> _domainEvents = new();
    0 references
    public IReadOnlyCollection<IDomainEvent> DomainEvents => _domainEvents;

    0 references
    protected void AddDomainEvent(IDomainEvent domainEvent) => _domainEvents.Add(domainEvent);
    0 references
    public void ClearDomainEvent(IDomainEvent domainEvent) => _domainEvents.Clear();

}
```



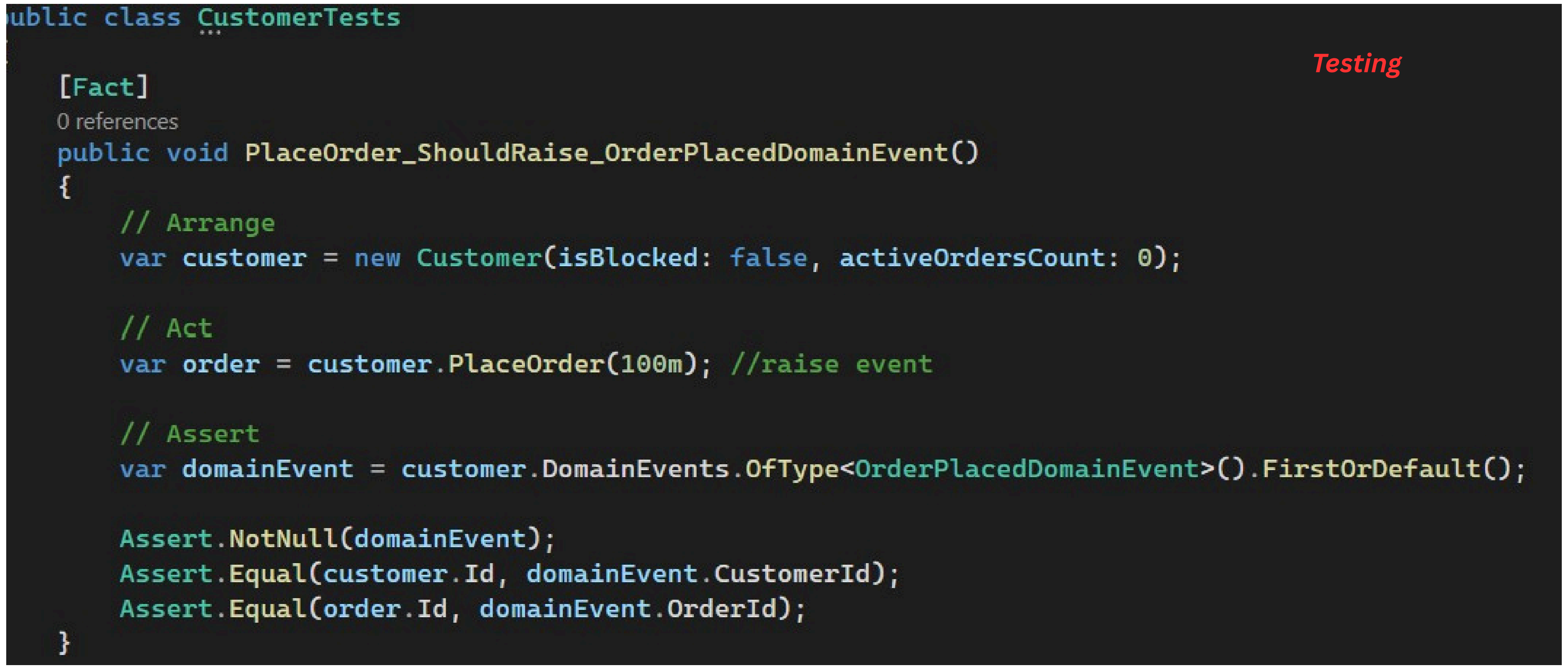
Entity

```csharp
public class Customer : AggregateRoot
{
    3 references
    public bool IsBlocked { get; private set; }
    2 references
    public int ActiveOrdersCount { get; private set; }

    3 references
    public Customer(bool isBlocked, int activeOrdersCount)
    {
        IsBlocked = isBlocked;
        ActiveOrdersCount = activeOrdersCount;
    }

    1 reference
    public Order PlaceOrder(decimal totalAmount)     //DOMAIN EVENT
    {
        if (IsBlocked)
            throw new InvalidOperationException("Blocked customer cannot place orders.");

        var order = new Order(Id, totalAmount, DateTime.UtcNow);
        ActiveOrdersCount++;

        var domainEvent = new OrderPlacedDomainEvent(Id, order.Id);
        AddDomainEvent(domainEvent);  //RAISING HERE

        return order;
    }
}
```



Domain Event

```csharp
public class OrderPlacedDomainEvent : IDomainEvent
{
    2 references
    public Guid CustomerId { get; }
    2 references
    public Guid OrderId { get; }
    0 references
    public DateTime OccurredOn { get; } = DateTime.UtcNow;

    1 reference
    public OrderPlacedDomainEvent(Guid customerId, Guid orderId)
    {
        CustomerId = customerId;
        OrderId = orderId;
    }
}
```



Testing

```csharp
ublic class CustomerTests

    [Fact]
    0 references
    public void PlaceOrder_ShouldRaise_OrderPlacedDomainEvent()
    {
        // Arrange
        var customer = new Customer(isBlocked: false, activeOrdersCount: 0);

        // Act
        var order = customer.PlaceOrder(100m); //raise event

        // Assert
        var domainEvent = customer.DomainEvents.OfType<OrderPlacedDomainEvent>().FirstOrDefault();

        Assert.NotNull(domainEvent);
        Assert.Equal(customer.Id, domainEvent.CustomerId);
        Assert.Equal(order.Id, domainEvent.OrderId);
    }
```