## Entites

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

```
ublic class Customer : AggregateRoot

    2 references
    public bool IsBlocked { get; private set; }
    1 reference
    public int ActiveOrdersCount { get; private set; }

    2 references
    public Customer(bool isBlocked, int activeOrdersCount)
    {
        IsBlocked = isBlocked;
        ActiveOrdersCount = activeOrdersCount;
    }
    //other logics for Customer
```

```
blic class Order : Entity

    1 reference
    public Guid CustomerId { get; private set; }
    1 reference
    public decimal TotalAmount { get; private set; }
    1 reference
    public DateTime OrderDate { get; private set; }

    0 references
    public Order(Guid customerId, decimal totalAmount, DateTime orderDate)
    {
        CustomerId = customerId;
        TotalAmount = totalAmount;
        OrderDate = orderDate;
    }

    //other behaviour of Order
```

## AR

```
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();

}
```

## Firuza Polad

## DomainService

```
public class CustomerOrderDomainService  //doesn't store any internal state
{
    private readonly ICustomerRepository _customerRepository;
    private readonly IOrderRepository _orderRepository;          //it uses repositories only to get data , no persistence logic

    3 references
    public CustomerOrderDomainService(ICustomerRepository customerRepository, IOrderRepository orderRepository)
    {
        _customerRepository = customerRepository;
        _orderRepository = orderRepository;
    }

    // this logic is not placed inside Customer or Order because it involves both entities,
    // will check if a customer can place a new order based on business rules.
    3 references
    public bool CanPlaceOrder(Guid customerId)
    {
        var customer = _customerRepository.GetById(customerId);

        if (customer is null)
            throw new ArgumentException("Customer does not exist");

        if (customer.IsBlocked)
            return false;

        int activeOrder = _orderRepository.CountActiveOrdersByCustomer(customerId);

        if (activeOrder >= 5)
            return false;

        return true;
    }
}
```

## Testing

```
public class CustomerOrderDomainServiceTests
{
    [Fact]
    0 references
    public void CanPlaceOrder_ReturnsTrue_WhenCustomerIsValidAndHasLessThan5Orders()
    {
        //Arrange
        var customerId = Guid.NewGuid();
        var customer = new Customer(isBlocked: false, activeOrdersCount: 3);

        var mockCustomerRepository = new Mock<ICustomerRepository>();
        mockCustomerRepository.Setup(p => p.GetById(customerId)).Returns(customer);

        var mockOrderRepository = new Mock<IOrderRepository>();
        mockOrderRepository.Setup(p => p.CountActiveOrdersByCustomer(customerId)).Returns(3);

        var service = new CustomerOrderDomainService(mockCustomerRepository.Object, mockOrderRepository.Object);

        //Act
        bool result = service.CanPlaceOrder(customerId);

        //Assert
        Assert.True(result);
    }

    [Fact]
    0 references
    public void CanPlaceOrder_ReturnsFalse_WhenCustomerHasTooManyOrders()
    {
        //Arrange
        var customerId = Guid.NewGuid();
        var customer = new Customer(isBlocked: false, activeOrdersCount: 6);

        var mockCustomerRepository = new Mock<ICustomerRepository>();
        mockCustomerRepository.Setup(p => p.GetById(customerId)).Returns(customer);

        var mockOrderRepository = new Mock<IOrderRepository>();
        mockOrderRepository.Setup(p => p.CountActiveOrdersByCustomer(customerId)).Returns(6);

        var service = new CustomerOrderDomainService(mockCustomerRepository.Object, mockOrderRepository.Object);

        //Act
        bool result = service.CanPlaceOrder(customerId);

        //Assert
        Assert.False(result);
    }
```

```
    [Fact]
    0 references
    public void CanPlaceOrder_ThrowsException_WhenCustomerNotFound()
    {
        //Arrange
        var customerId = Guid.NewGuid();

        var mockCustomerRepo = new Mock<ICustomerRepository>();
        mockCustomerRepo.Setup(r => r.GetById(customerId)).Returns((Customer)null);

        var mockOrderRepo = new Mock<IOrderRepository>();
        var service = new CustomerOrderDomainService(mockCustomerRepo.Object, mockOrderRepo.Object);

        //Act
        Action act = () => service.CanPlaceOrder(customerId);

        //Assert
        Assert.Throws<ArgumentException>(act);
    }
```