Practical Domain-Driven Design With EF Core

Hossam Barakat

Technical Lead at Willow

@hossambarakat_ | www.hossambarakat.net

What is Domain-Driven Design?

"Domain-Driven Design is an approach to software development that centers the development on programming a ***domain model*** that has a rich understanding of the processes and rules of a domain. " -- Martin Fowler

What is Domain Model?

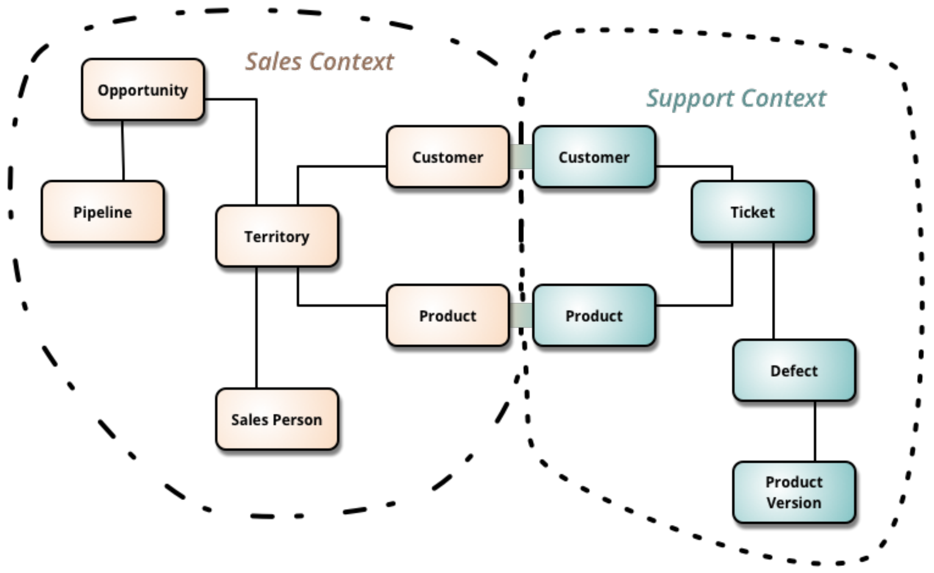
“An object model of the domain that incorporates both ***behavior*** and ***data***. ” - Martin Fowler

“A system of abstractions that describes ***selected*** aspects of a domain and can be used to solve problems related to that domain. ” -- Eric Evans

Strategic Design

Bounded Contexts

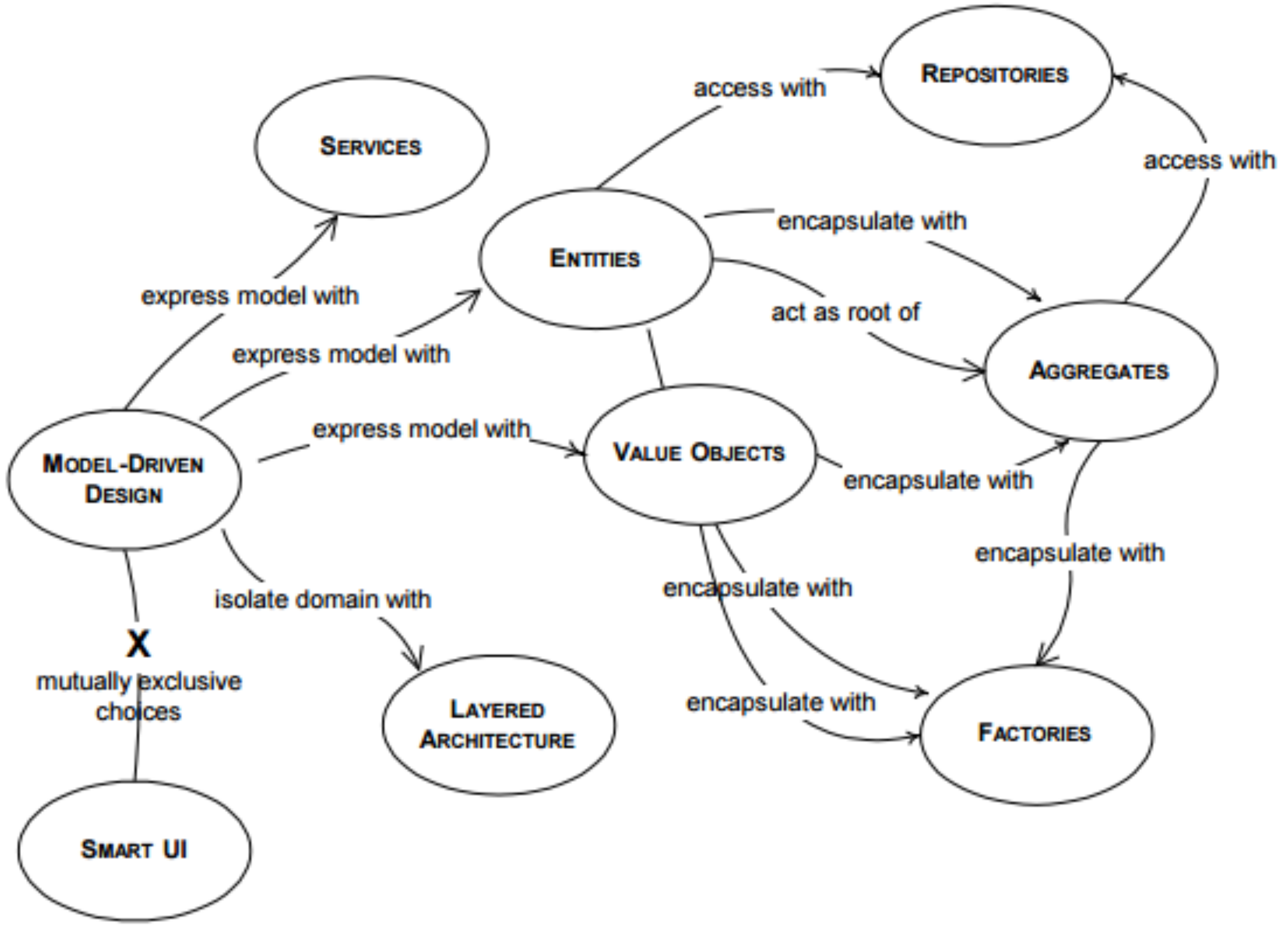
- A defined part of software where particular terms, definitions and rules apply in a consistent way
- Clear boundaries between different parts of the system



https://martinfowler.com/bliki/BoundedContext.html

Focus on the Core Domain

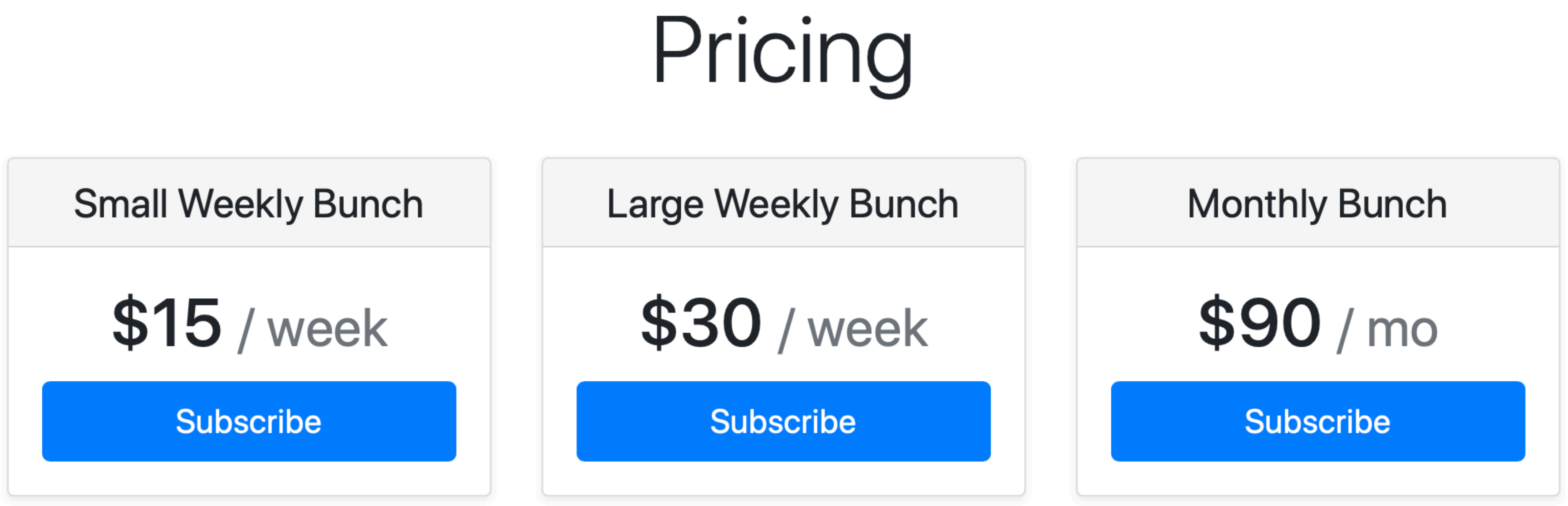
Inside the Core Domain



How to apply DDD to my legacy codebase?

Sample Domain

Sample Domain

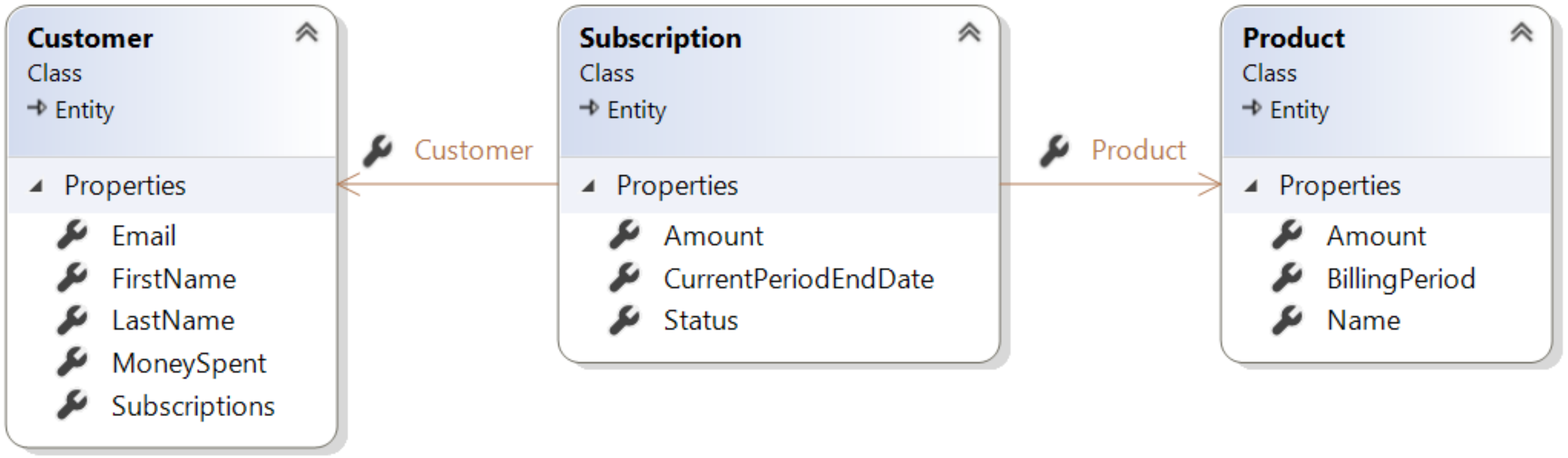


Pricing

Small Weekly Bunch	Large Weekly Bunch	Monthly Bunch
$15 / week	$30 / week	$90 / mo
Subscribe	Subscribe	Subscribe

@hossambarakat_

Domain Model



Demo

Extract Methods

```csharp
var customer = await _subscriptionContext
    .Customers
    .Include(x=>x.Subscriptions)
    .FirstAsync(x=> x.Id == request.CustomerId, cancellationToken: cancellationToken);

var product = await _subscriptionContext.Products.FindAsync(request.ProductId);

var subscriptionAmount = product.Amount;
if (customer.MoneySpent >= 100)
{
    subscriptionAmount *= 0.8M;
}
else if (customer.MoneySpent >= 1000)
{
    subscriptionAmount *= 0.5M;
}

var currentPeriodEndDate = product.BillingPeriod switch
{
    BillingPeriod.Weekly => DateTime.UtcNow.AddDays(7),
    BillingPeriod.Monthly => DateTime.UtcNow.AddMonths(1),
    _ => throw new InvalidOperationException()
};

var subscription = new Subscription
{
    Id = Guid.NewGuid(),
    Customer = customer,
    Product = product,
    Amount = subscriptionAmount,
    Status = SubscriptionStatus.Active,
    CurrentPeriodEndDate = currentPeriodEndDate
};
customer.Subscriptions.Add(subscription);
customer.MoneySpent += subscription.Amount;

await _subscriptionContext.SaveChangesAsync(cancellationToken);

await _emailSender.SendEmailAsync("Congratulations! You subscribed to a cool product");
return Unit.Value;
```

```csharp
var customer = await _subscriptionContext
    .Customers
    .Include(x=>x.Subscriptions)
    .FirstAsync(x=> x.Id == request.CustomerId, cancellationToken: cancellationToken);

var product = await _subscriptionContext.Products.FindAsync(request.ProductId);

var subscriptionAmount = CalculateSubscriptionAmount(product, customer);

var currentPeriodEndDate = CalculateCurrentPeriodEndDate(product);

AddSubscriptionToCustomer(customer, product, subscriptionAmount, currentPeriodEndDate);

await _subscriptionContext.SaveChangesAsync(cancellationToken);

await _emailSender.SendEmailAsync("Congratulations! You subscribed to a cool product");
return Unit.Value;
```

@hossambarakat_

Avoid Public Setters

```csharp
public class Subscription : Entity
{
    public SubscriptionStatus Status { get; set; }
    public Customer Customer { get; set; }
    public Product Product { get; set; }
    public decimal Amount { get; set; }
    public DateTime CurrentPeriodEndDate { get; set; }
}
```

```csharp
public class Subscription : Entity
{
    public Subscription(Customer customer, Product product,
            decimal amount, DateTime currentPeriodEndDate)
    {
        Id = Guid.NewGuid();
        Customer = customer ?? throw new ArgumentNullException(nameof(customer));
        Product = product ?? throw new ArgumentNullException(nameof(product));
        Amount = amount >= 0m ? amount : throw new ArgumentOutOfRangeException(nameof(amount));
        CurrentPeriodEndDate = currentPeriodEndDate;
        Status = SubscriptionStatus.Active;
    }

    public Customer Customer { get; private set; }
    public Product Product { get; private set; }
    public decimal Amount { get; private set; }
    public DateTime CurrentPeriodEndDate { get; private set; }
    public SubscriptionStatus Status { get; private set; }
}
```

Encapsulate Behavior in Domain Model

```
public async Task<Unit> Handle(SubscribeRequest request, CancellationToken cancellationToken)
{
    ....

    var subscription = new Subscription
    {
        Id = Guid.NewGuid(),
        Customer = customer,
        Product = product,
        Amount = subscriptionAmount,
        Status = SubscriptionStatus.Active,
        CurrentPeriodEndDate = currentPeriodEndDate
    };
    customer.Subscriptions.Add(subscription);
    customer.MoneySpent += subscription.Amount;

    ...
}
```

```
public class Customer: Entity
{
    public string Email { get; set;}
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public decimal MoneySpent { get; set; }

    private List<Subscription> _subscriptions = new List<Subscription>();
    public IReadOnlyCollection<Subscription> Subscriptions => _subscriptions.AsReadOnly();

    public void AddSubscription(Product product, decimal subscriptionAmount,
        DateTime currentPeriodEndDate)
    {
        var subscription = new Subscription(this, product, subscriptionAmount, currentPeriodEndDate);
        _subscriptions.Add(subscription);
        MoneySpent += subscription.Amount;
    }
}
```

@hossambarakat_

Encapsulate Collection

```
public decimal MoneySpent { get; set; }
public List<Subscription> Subscriptions { get; set; }
```

```
public decimal MoneySpent { get; private set; }
public List<Subscription> _subscriptions = new List<Subscription>();
public IReadOnlyCollection<Subscription> Subscriptions => _subscriptions.AsReadOnly();
```

@hossambarakat_

Encapsulate Behavior in Domain Model

```csharp
var currentPeriodEndDate = product.BillingPeriod switch
{
    BillingPeriod.Weekly => DateTime.UtcNow.AddDays(7),
    BillingPeriod.Monthly => DateTime.UtcNow.AddMonths(1),
    _ => throw new InvalidOperationException()
};
```

```csharp
public class Product: Entity
{
    public string Name { get; set;}
    public decimal Amount { get; set; }
    public BillingPeriod BillingPeriod { get; set; }

    public DateTime CalculateCurrentPeriodEndDate()
    {
        var currentPeriodEndDate = BillingPeriod switch
        {
            BillingPeriod.Weekly => DateTime.UtcNow.AddDays(7),
            BillingPeriod.Monthly => DateTime.UtcNow.AddMonths(1),
            _ => throw new InvalidOperationException()
        };
        return currentPeriodEndDate;
    }
}
```

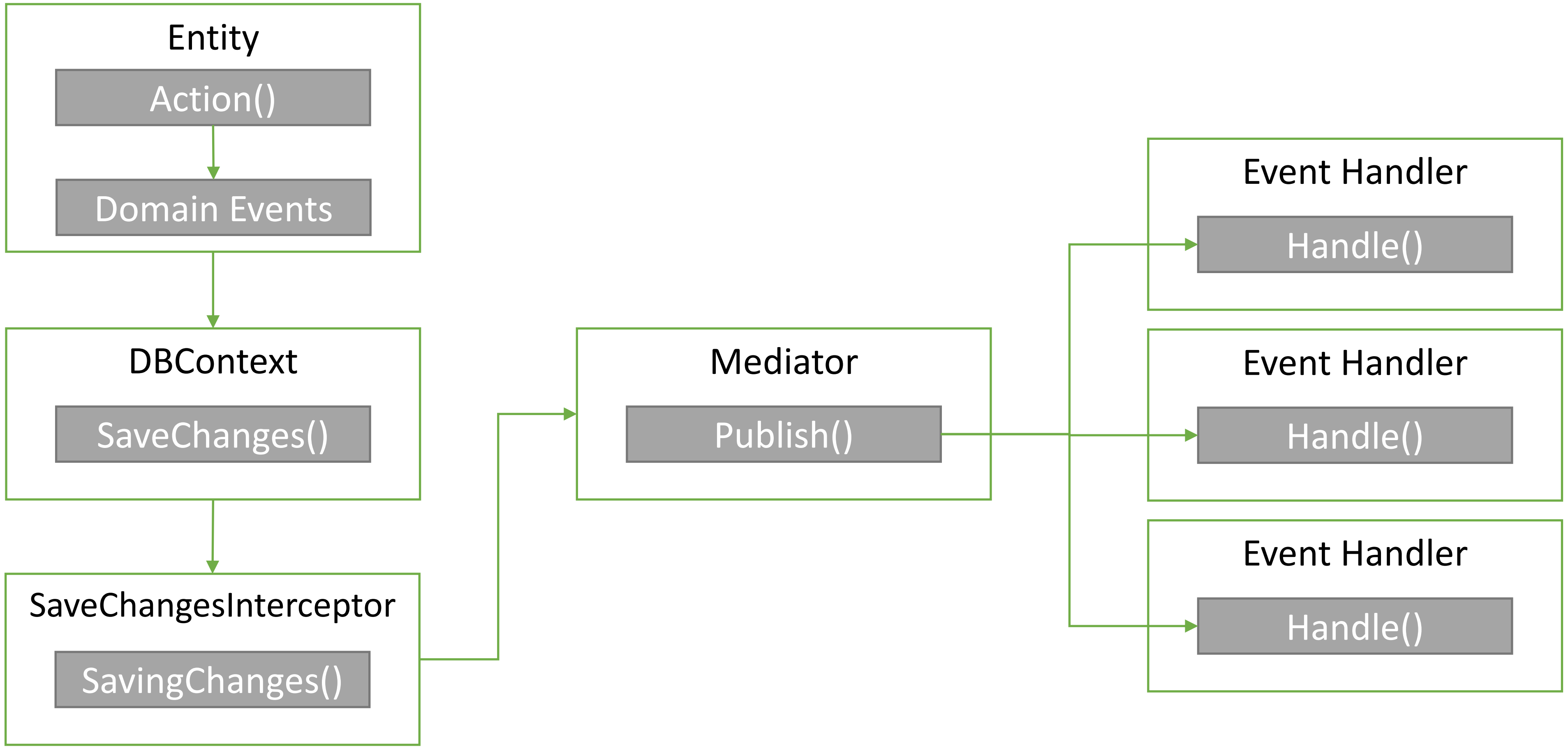
Encapsulate Behavior in Domain Service

```
public interface ISubscriptionAmountCalculator
{
    decimal CalculateSubscriptionAmount(Product product, Customer customer);
}
public class SubscriptionAmountCalculator : ISubscriptionAmountCalculator
{
    public decimal CalculateSubscriptionAmount(Product product, Customer customer)
    {
        var subscriptionAmount = product.Amount;
        if (customer.MoneySpent >= 100)
        {
            subscriptionAmount *= 0.8M;
        }
        else if (customer.MoneySpent >= 1000)
        {
            subscriptionAmount *= 0.5M;
        }

        return subscriptionAmount;
    }
}
```

Double Dispatch

```
public void AddSubscriptionToCustomer(Product product, ISubscriptionAmountCalculator subscriptionAmountCalculator)
{
    var subscriptionAmount = subscriptionAmountCalculator.CalculateSubscriptionAmount(product, this);
    var subscription = new Subscription(this, product, subscriptionAmount);

    _subscriptions.Add(subscription);
    MoneySpent += subscription.Amount;
}
```

Domain Events



From Primitive Obsession to Value Objects

```csharp
public class Customer: Entity
{
    public string Email { get; set;}
    public string FirstName { get; set; }
    public string LastName { get; set; }
    ...
}
```

```csharp
public record CustomerName(string FirstName, string LastName);

public class Customer: Entity
{
    public Customer(string email, CustomerName customerName)
    {
        Id = Guid.NewGuid();
        Email = email ?? throw new ArgumentNullException(nameof(email));
        CustomerName = customerName ?? throw new ArgumentNullException(nameof(customerName));
    }

    public CustomerName CustomerName { get; private set; }
    ...
}
```

```csharp
builder.OwnsOne(x => x.CustomerName, nameBuilder =>
{
    nameBuilder.Property(p => p.FirstName)
        .HasColumnName("FirstName")
        .IsRequired();
    nameBuilder.Property(p => p.LastName)
        .HasColumnName("LastName")
        .IsRequired();
});
builder.Navigation(e => e.CustomerName).IsRequired();
```

@hossambarakat_

Where is the Repository?

```
public class CustomerRepository : ICustomerRepository, IDisposable
{
    private SubscriptionContext context;

    public CustomerRepository(SubscriptionContext context)
    {
        this.context = context;
    }

    public IEnumerable<Customer> GetCustomers()
    {
        return context.Customers.ToList();
    }

    public Customer GetCustomerByID(Guid id)
    {
        return context.Customers.Find(id);
    }

    public void Save()
    {
        context.SaveChanges();
    }

}
```

Can the Unit of Work help us?

```
public class UnitOfWork
{
    private readonly SubscriptionContext _context;

    public UnitOfWork(SubscriptionContext context)
    {
        _context = context;
    }

    public void Save()
    {
        _context.SaveChanges();
    }
}
```

Embrace the DbContext

```
public class SubscriptionContext : DbContext, ISubscriptionContext
{
    public SubscriptionContext(DbContextOptions<SubscriptionContext> options)
        : base(options)
    {
    }

    public DbSet<Comment> Comments { get; set; }
    public DbSet<Invoice> Invoices { get; set; }
    public DbSet<InvoiceLine> InvoiceLines { get; set; }

    private IDbContextTransaction _transaction;

    public void BeginTransaction()
    {
        _transaction = Database.BeginTransaction();
    }

    public void Commit()
    {
        try
        {
            SaveChanges();
            _transaction.Commit();
        }
        finally
        {
            _transaction.Dispose();
        }
    }

    public void Rollback()
    {
        _transaction.Rollback();
        _transaction.Dispose();
    }
}
```

@hossambarakat_

What about custom queries?

- Query classes
- Specification pattern
- Extension Methods

Specification: Current Query

```
var queryResult = await _context.Subscriptions
    .Where(s => s.Status == SubscriptionStatus.Active)
    .Where(s => s.Customer.Id == request.CustomerId)
    .Select(x=> new GetActiveSubscriptionsResponse
    {
        ProductName = x.Product.Name,
        BillingPeriod = x.Product.BillingPeriod.ToString()
    })
    .ToListAsync(cancellationToken);
```

Specification: Base Classes

```
public interface ISpecification<T>
{

    public Expression<Func<T,bool>> Criteria { get; set; }

    public List<Expression<Func<T, object>>> Includes { get; }

    public List<string> IncludeStrings { get; }
}


public abstract class BaseSpecification<T>: ISpecification<T>
{

    public Expression<Func<T,bool>> Criteria { get; set; }

    public List<Expression<Func<T, object>>> Includes { get; } = new();

    public List<string> IncludeStrings { get; } = new();

    protected virtual void AddInclude(Expression<Func<T, object>> includeExpression)
    {
        Includes.Add(includeExpression);
    }


    protected virtual void AddInclude(string includeString)
    {
        IncludeStrings.Add(includeString);
    }
}
```

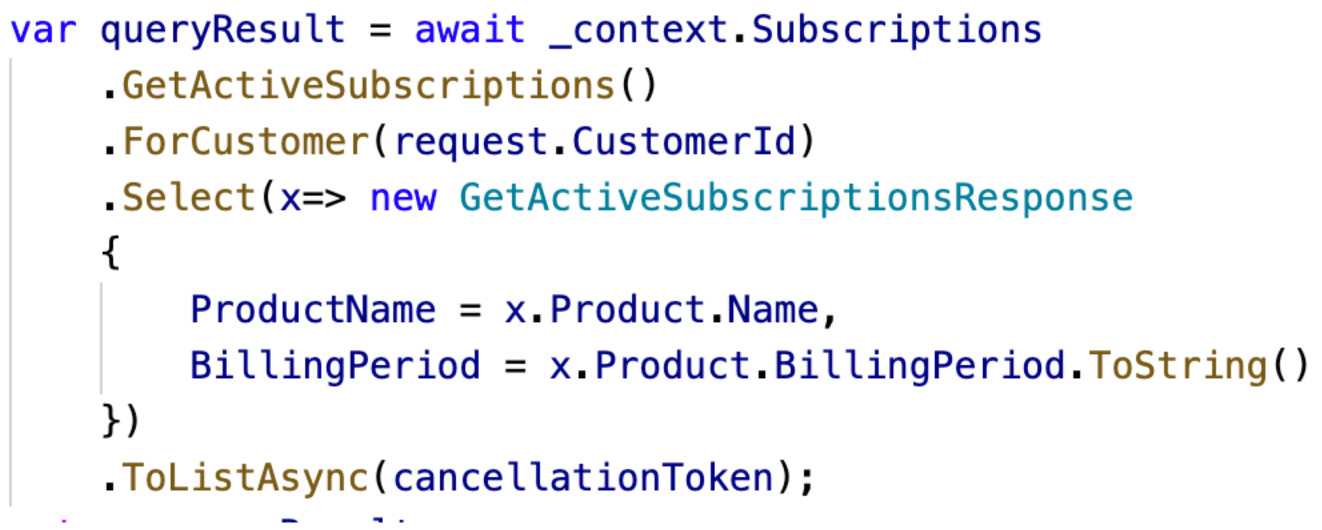
Specification: Concrete Sample

```
public class ActiveSubscriptionSpecification : BaseSpecification<Subscription>
{

    public ActiveSubscriptionSpecification()
    {
        Criteria = s => s.Status == SubscriptionStatus.Active;
    }
}

public class CustomerSubscriptionsSpecification : BaseSpecification<Subscription>
{

    public CustomerSubscriptionsSpecification(Guid customerId)
    {
        Criteria = s => s.Customer.Id == customerId;
    }
}
```

Specification: Concrete Sample

```
var queryResult = await _context.Subscriptions
    .Where(new ActiveSubscriptionSpecification())
    .Where(new CustomerSubscriptionsSpecification(request.CustomerId))
    .Select(x=> new GetActiveSubscriptionsResponse
    {
        ProductName = x.Product.Name,
        BillingPeriod = x.Product.BillingPeriod.ToString()
    })
    .ToListAsync(cancellationToken);
```

Specification: Extension Methods



```
var queryResult = await _context.Subscriptions
    .GetActiveSubscriptions()
    .ForCustomer(request.CustomerId)
    .Select(x=> new GetActiveSubscriptionsResponse
    {
        ProductName = x.Product.Name,
        BillingPeriod = x.Product.BillingPeriod.ToString()
    })
    .ToListAsync(cancellationToken);
```

@hossambarakat_

Base Entity ID

```
public abstract class Entity
{
    public Guid Id { get; set; }
}
```

```
public abstract class Entity<TIdentity>
{
    public TIdentity Id { get; set; }
}
```

```
public class SubscriptionId : Identity
{
    public SubscriptionId(Guid value) : base(value)
    {
    }
}
```

Resources

- https://github.com/hossambarakat/Subscriptions-DDD
- https://github.com/ardalis/Specification

Questions

Thanks

Hossam Barakat

Tech Lead at Willow

@HossamBarakat_