ZAN KAVTASKIN

Musings about Software Engineering

Tuesday, 12 November 2013

Applied Domain-Driven Design (DDD), Part 5 - Domain Service

Domain Service is not be confused with Application Service or Web Service. Domain Service lives in the Domain Model Layer. Unlike Application or Web Service, Domain Service should not be called each time to access Domain Model Layer. You can call your repository interface in the Application Layer and get the Domain Entity directly.

Requirement:

- Your business is responsible for collecting and paying Value Added Tax (VAT) based off your business location, your customer's location, and the type of product you are selling.

Domain Service Sample:

```
public class TaxDomainService : ITaxDomainService
    {
        readonly IRepository<ProductTax> productTax;
        readonly IRepository<CountryTax> countryTax;
        readonly Settings settings;

        public TaxDomainService(Settings settings, IRepository<ProductTax> productTax, IRepository<CountryTax> countryTax)
        {
            this.productTax = productTax;
            this.countryTax = countryTax;
            this.settings = settings;
        }

        public decimal Calculate(Customer customer, Product product)
        {
            CountryTax customerCountryTax = this.countryTax.FindById(customer.Country.Id);
            CountryTax businessCountryTax = this.countryTax.FindById(settings.BusinessCountry.Id);
            ProductTax productTax = this.productTax.FindById(product.Code.Id);
            return (product.Cost * customerCountryTax.Percentage) +
                (product.Cost * businessCountryTax.Percentage) + (product.Cost * productTax.Percentage);
        }
```

```
}
```

Example usage:

```
Customer customer = this.repositoryCustomer.FindById(customerId);
Product product = this.repositoryProduct.FindById(productId);

decimal tax = this.taxDomainService.Calculate(customer, product);

customer.Cart.Add(CartProduct.Create(customer.Cart, product, productQuantity, tax));
```

Would like to see full working example?
Browse "Domain-Driven Design Example" Repository On Github

Summary:

- Domain Service allows you to capture logic that doesn't belong in the Domain Entity.
- Domain Service allows you to orchestrate between different Domain Entities.

Tips:

- Don't create too many Domain Services, most of the logic should reside in the domain entities, event handlers, etc.
- It's a great place for calculation and validation as it can access entities, and other kind of objects (e.g. Settings) that are not available via the entity graph.
- Methods should return primitive types, custom enums are fine too.

**Note: Code in this article is not production ready and is used for prototyping purposes only. If you have suggestions or feedback please do comment.*

-
-
-
-
-

Rate this blog post (21 Votes)

Posted by Zan Kavtaskin
Labels: domain-driven design, software engineering

3 comments:

Blog Archive

- ▶ 2020 (2)
- ▶ 2019 (1)
- ▶ 2018 (7)
- ▶ 2017 (5)
- ▶ 2016 (9)
- ▶ 2014 (3)
- ▼ 2013 (9)
  - ▶ Dec (3)
  - ▼ Nov (3)
    - Applied Domain-Driven Design (DDD), Part 6 - Appli...
    - Applied Domain-Driven Design (DDD), Part 4 - Infra...
    - Applied Domain-Driven Design (DDD), Part 5 - Domai...
  - ▶ Oct (1)
  - ▶ Sep (2)

Unknown 8 November 2017 at 00:41

This comment has been removed by a blog administrator.

Reply

Unknown 5 July 2018 at 02:59

How about calling the SERVICE class CalculateTax instead?

Add some modifiers setCustomer(customer) and setProduct(product), and then a function getTax().

Maybe?

Reply

Anonymous 2 August 2018 at 03:06

Hi,
...could a DomainService also implement CRUD functionality for a domain model objects? I saw some examples like a ProductService which encapsulated createProduct(), getProduct(), and so on. Even the product collection was placed in the ProductService. Its structure was very similar to a ProductRepository. I am confused now.

Reply

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)