ZAN KAVTASKIN

Musings about Software Engineering

Wednesday, 2 October 2013

Applied Domain-Driven Design (DDD), Part 3 - Specification Pattern

Specification pattern is great, David Fancher wrote a great piece on it, i suggest you read it before you continue.

In short, specification pattern allows you to chain business queries.

Example:

```
            ISpecification<Customer> spec =
                new CustomerRegisteredInTheLastDays(30).And(new CustomerPurchasedNumOfProducts(2));
```

Entity from previous posts in this series:

```
 public class Customer : IDomainEntity
    {
        private List<Purchase> purchases = new List<Purchase>();

        public Guid Id { get; protected set; }
        public string FirstName { get; protected set; }
        public string LastName { get; protected set; }
        public string Email { get; protected set; }
        public string Password { get; protected set; }
        public DateTime Created { get; protected set; }
        public bool Active { get; protected set; }

        public ReadOnlyCollection<Purchase> Purchases { get { return this.purchases.AsReadOnly(); } }

        public static Customer Create(string firstname, string lastname, string email)
        {
            Customer customer = new Customer()
            {
                FirstName = firstname,
```

Search This Blog

Search

About Me

Zan Kavtaskin

Nottingham, United Kingdom

I am a Software Director, Architect and Engineer. I work at MHR as a Software Delivery Director and I have also written software for companies such as Experian, Emirates and Royal Mail.

View my complete profile

Popular Posts

- Applied Domain-Driven Design (DDD), Part 1 - Basics
- Applied Domain-Driven Design (DDD), Part 0 - Requirements and Modelling
- Applied Domain-Driven Design (DDD), Part 2 - Domain Events
- Applied Domain-Driven Design (DDD), Part 3 - Specification Pattern
- Applied Domain-Driven Design (DDD), Part 4 - Infrastructure
- Applied Domain-Driven Design (DDD), Part 6 - Application Services
- Applied Domain-Driven Design (DDD), Part 5 - Domain Service
- Applied Domain-Driven Design (DDD), Part 7 - Read Model
- Applied Domain-Driven Design (DDD) - Event Logging & Sourcing For Auditing
- Unit Of Work Abstraction For NHibernate or Entity Framework C# Example

```
                LastName = lastname,
                Email = email,
                Active = true,
                Created = DateTime.Today
            };

            DomainEvents.Raise<CustomerCreated>(new CustomerCreated() { Customer = customer });
            return customer;
        }

        public Purchase Checkout(Cart cart)
        {
            Purchase purchase = Purchase.Create(this, cart.Products);
            this.purchases.Add(purchase);
            DomainEvents.Raise<CustomerCheckedOut>(new CustomerCheckedOut() { Purchase = purchase });
            return purchase;
        }
    }
```

Specification Examples:

```
public class CustomerRegisteredInTheLastDays : SpecificationBase<Customer>
    {
        readonly int nDays;

        public CustomerRegisteredInTheLastDays(int nDays)
        {
            this.nDays = nDays;
        }

        public override Expression<Func<Customer,bool>>  SpecExpression
        {
            get
            {
                return customer => customer.Created >= DateTime.Today.AddDays(-nDays)
                    && customer.Active;
            }
        }
    }

    public class CustomerPurchasedNumOfProducts : SpecificationBase<Customer>
    {
        readonly int nPurchases;

        public CustomerPurchasedNumOfProducts(int nPurchases)
        {
            this.nPurchases = nPurchases;
        }
        public override Expression<Func<Customer,bool>>  SpecExpression
        {
            get
            {
                return customer => customer.Purchases.Count == this.nPurchases
                    && customer.Active;
            }
        }
    }
```

Abstract Repository Query Example:

```
            IRepository<Customer> customerRepository = new Repository<Customer>();

            ISpecification<Customer> spec =
                new CustomerRegisteredInTheLastDays(30).And(new CustomerPurchasedNumOfProducts(2));

            IEnumerable<Customer> customers = customerRepository.Find(spec);
```

Abstract Repository Example:

```
    public interface IRepository<TEntity>
        where TEntity : IDomainEntity
    {
        TEntity FindById(Guid id);
        TEntity FindOne(ISpecification<TEntity> spec);
        IEnumerable<TEntity> Find(ISpecification<TEntity> spec);
        void Add(TEntity entity);
        void Remove(TEntity entity);
    }
```

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

Summary:

- Specification allows you to query data in a abstract way i.e. you can query memory collections or an RDBMS. This ensures persistence/infrastructure ignorance.
- Specification encapsulates a business rule in one spec.
- Specification pattern allows you to chain your business rules up.
- Specification makes your domain layer DRY i.e. you don't need to write same LINQ all over again.
- Specifications are easy to unit test.
- Specifications are stored in the domain layer, this provides full visibility.

- Specifications are super elegant.

Tips:

- Break complex business logic rules down in your specification as NHibernate might struggle to interpret them in to a SQL query. This is a generally good tip as you don't want messy SQL hitting your database.
- Query data around the entity properties, don't try and change the properties on the entity i.e. instead of writing customer.Created.AddDays(30) >= DateTime.Today write customer.Created >= DateTime.Today.AddDays(-30). The former will try and compile it as a SQL and will fail as it's too complex, the latter will convert the value to a parameter.
- As specifications are logical queries they should not change state of the caller or the calling objects. i.e. don't call state changing methods, such as customer.Checkout(....) && customer.Active == true. This tip goes hand in hand with the tip above.

Useful links:

- Specifications, Expression Trees, and NHibernate a fantastic article with great examples on how to use spefifications with NHibernate.
- Specification Pattern, basic explanation of boolean specification pattern.

**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.*

No comments:

Post a Comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)