Value objects,

EMOCON 2017 S/S

도메인 주도 개발의 처음과 끝

차영호

@andrewchaa
andrew.yh.chaa@gmail.com
http://andrewchaa.me.uk

Who am I?

- Totaljobs.com -> Barclays -> Huddle -> MarketInvoice -> GSA Capital -> Trenbe
- 지금은 Trenbe, 유럽의 명품 브랜드를 한중일에 판매하는 스타트업
- 평범한 C# 개발자입니다
- 그런데 웬지 자바스크립트가 좋아요.

- www.andrewchaa.me.uk
- @andrewchaa

Primitive Type 중독

With Primitives!

Primitives

- 사실 어떤 언어로 개발하든 기본형은 말 그대로 기본 (building blocks)이 됩니다

- strings, int, decimal, float, booleans, …

```
public class Address {
    public string PostCode { get; set; }
}
```

그런데 …

- 정말 string으로 우편 번호를 표현할 수 있는지?

Post Code	String
8 Character limit including space	As long as database allows
Only Alpha-Numeric	Allow • ( ) £ ! * + = • •  • Even null and empty string

```
public class PostCode {
    private readonly string _value;
    public PostCode (string value) {
        _value = value;
    }
    public string Value {
        get { return _value; }
    }
}
```

Primitives 좋아하다 골로 갑니당

- User story 3971 “bid as little as 0.1% value of a trade”
- int -> decimal

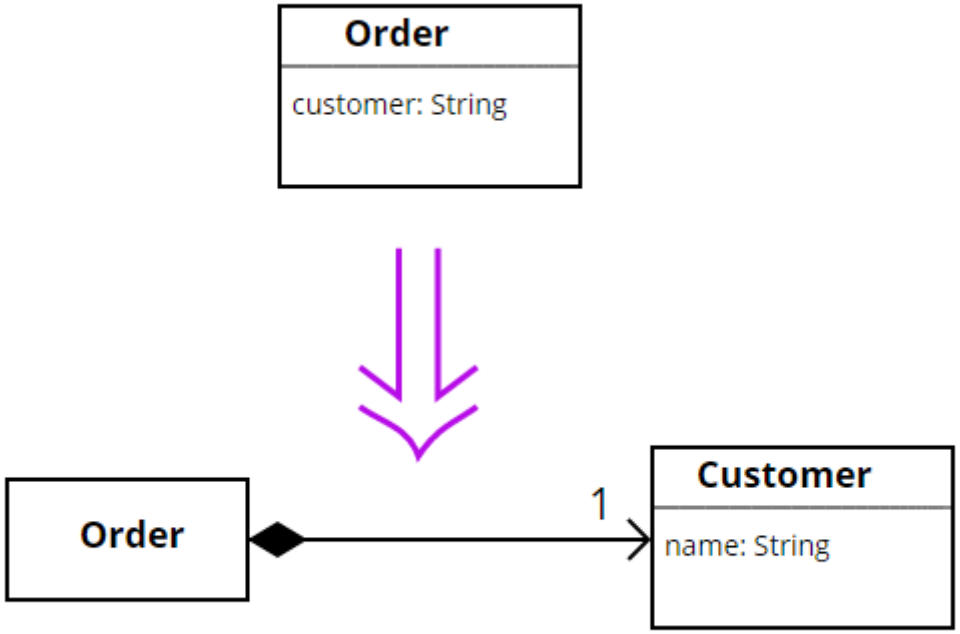
- Over 100 places to change
- Estimated to 32.5 hours of work!

Fowler 옹의 말씀…

As the system matures

Replace Data Value with Object

- by Martin Fowler



Why we prefer value objects

Business Logic

string data type can’t handle the business logic in the Post Code.
It accepts any text, even empty string!

Validation

```
public static bool IsValid(string candidate)
{
    if (string.IsNullOrEmpty(candidate))
        return false;
    …
    …
     return true;
}
```

Value objects have behaviours

- Often quite a lot!

Parsing

```
public static bool TryParse(string candidate, out PostCode postCode)
{
    postCode = null;
    if (string.IsNullOrWhiteSpace(candidate))
        return false;
     ….
    postCode = new PostCode(candidate.split(“ ”)[0],
        candidate.split(“”)[1]);
    return true;
}
```

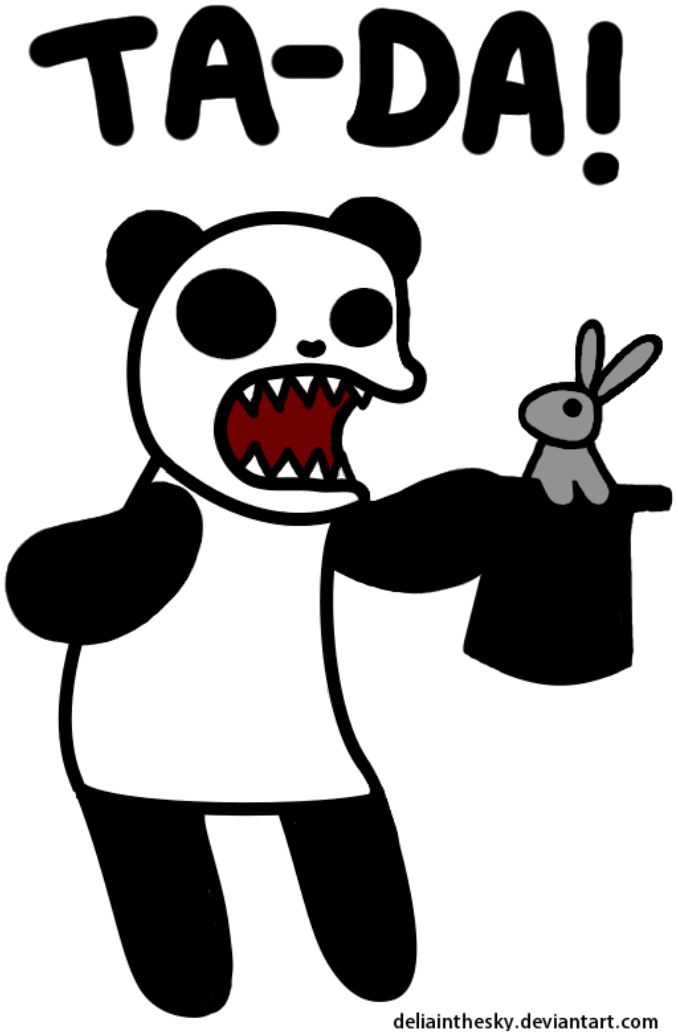
Slim and focused Domain objects

Wrapping primitive types in F#

```
type EmailAddress = EmailAddress of string
type PostCode = PostCode of string
type CountyCode = CountyCode of string
```

Are we convinced now? (약이 좀 팔렸는지?)

Value Objects



Describe something about the entity or the things it owns

- Ship -> Cargo capacity
- Grocery -> Stock level
- Financial Report -> Quarterly turnovers

Represent a descriptive, identity-less concept

- I pay £3.99 for Brie, Avaocado, and Tomato toast at Pret
- As long as the value is £3.99, they don’t care which coins I use

```
public class BankAccount {
  public BankAccount(Guid id, Money startingBalance) {
    this.Id = id;
    this.Balance = startingBalance;
  }

  public Guid Id { get; private set; }
  public Money Balance { get; private set; } .
  ..
}
```

Value object upon value objects

```
public class Money {
  public Money(int amount, Currency currency) {
    Amount = amount;
    Currency = currency;
  }

  private int Amount { get; set; }
  private Currency Currency { get; set; }
  ...
}
```

Enhance Explicitness

- When price is
- between 0.01 and 0.99: increase bid by 0.05
- between 1.00 and 0.20: increase bid by 0.20

Price as primitives

```
public class WinningBid {
  ...
  public int Price { get; private set; }
  ...
}
```

The logic goes into a domain object or service.

```
public class Price {
  …
  public Money Amount { get; private set; }

  public Money IncreseBid () {
    if (Amount.IsGreaterThan(new Money(0.01m)) &&
         Amount.IsLessThanOrEqualTo(new Money(0.99m)))
        return Amount.add(new Money(0.05m));
    …
}
```

```
public class Price {
  …
  public Money Amount { get; private set; }

  public Money IncreaseBid() {

    …
    if (Amount.IsGreaterThan(new Money(1.00m)) &&
        Amount.IsLessEqualTo(new Money(4.99m)))
       return Amount.add(new Money(0.20m));
  }
```

- Now the value object handles the business logic

More behaviours into value objects

- To keep entities slim and focused

Characteristics of value objects

Identity-less

- May have ids using some database persistence strategies

Attribute-based Equality

- Considered equal if they have the same value

- In case of money,
- this.Value == other.Value
- this.Currency == other.Currency

Behaviour-rich and Cohesive

- Exposes expressive domain-oriented behaviours in the same place

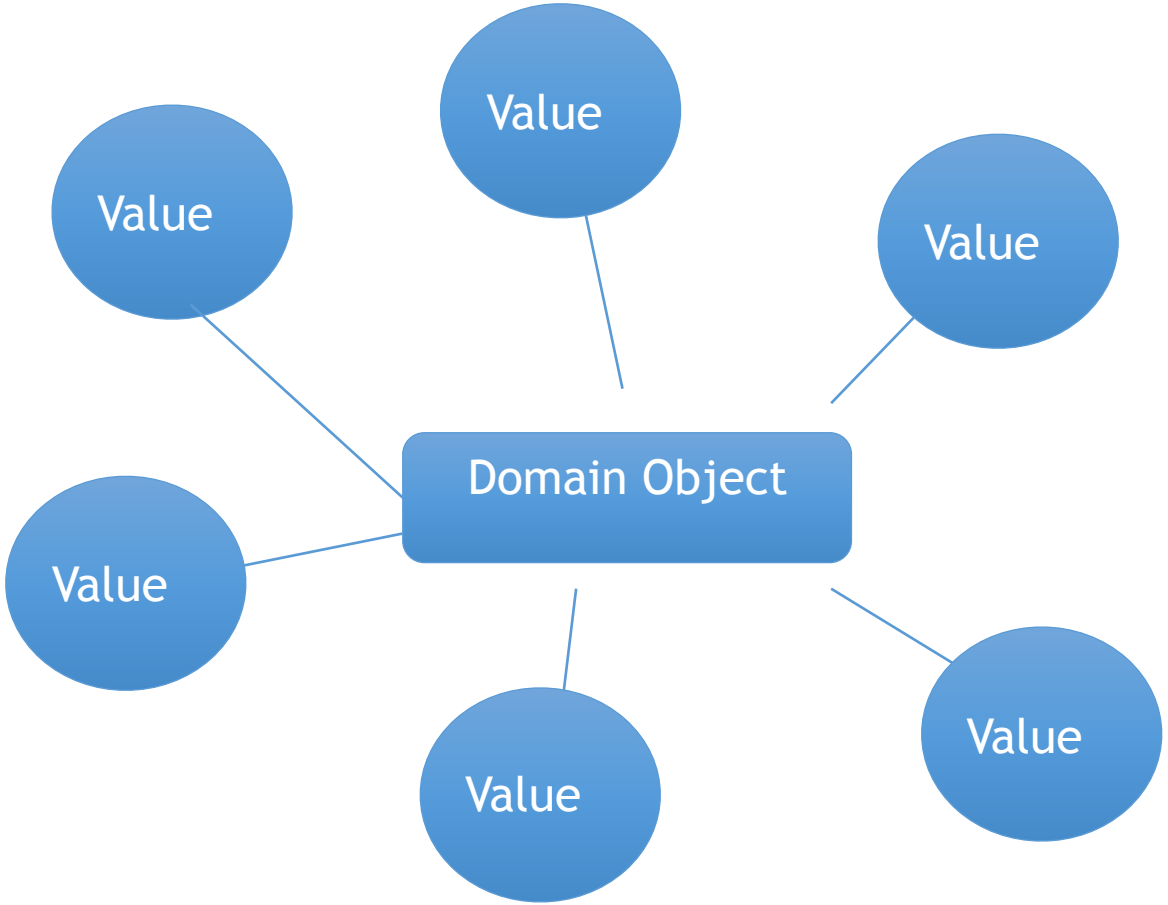
```
public class Meters {

  public Miles ToMiles() {
    return new Miles(DistanceInMeters * 0.000621371m);
  }

  public Kilometers ToKilometers() {
    return new Kilometers(DistanceInMeters / 1000m);
  }

}
```

Slim Domain object with Rich Value Objects



Immutable

- Use readonly instance variables

```
public class Money : ValueObject<Money> {
  private readonly decimal Value;

  public Money() : this(0m) { }
  public Money(decimal value) { Value = value; }

  public Money Add(Money money) {
    return new Money(Value + money.Value);
  }

}
```

Combinable

```
public class Money : ValueObject<Money> {

  public Money Add(Money money) {
    return new Money(Value + money.Value);
  }

  public static Money operator + (Money left, Money right) {
    return new Money(left.Value + right.Value);
  }
}
```

Self-Validating

- All money is accurate to two decimal places
- All money must be a positive value
  - Balance can be a negative value

Validate in the constructor

```
public Money(decimal value)
{
    Validate(value);
    Value = value;
}

private void Validate(decimal value)
{
    if (value % 0.01m != 0)
      throw new  MoreThanTwoDecimalPlacesException();
    if (value < 0) throw new CannotBeANegativeValueException();
}
```

Validate inside a factory method

```
public class Money : ValueObject<Money> {
  // ..

  public static Money Create(decimal amount)  {
    if (amount % 0.01m != 0)
      throw new MoreThanTwoDecimalPlacesException();

    if (amount < 0)
      throw new CannotBeANegativeValueException();

    return new Money(amount);
  }
}
```

Testable

```
public void First_names_cannot_be_empty() {
  try    {
    var name = new Name("","Torvalds");
  }    catch (ApplicationException e)    {
    Assert.AreEqual("You must specify a first name.", e.Message);
    return;
  }

   Assert.Fail("No ApplicationException was thrown");
}
```

Common Modeling Patterns

Static Factory Methods

- Free to choose
- Ignore as you wish

```
public class Height {
  …
  public static Height FromFeet(int feet) {
    return new Height(feet, MeasurmentUnit.Feet);
  }

  public static Height FromMetres(int metres) {
    return new Height(metres, MeasurmentUnit.Metres);
  }
}
```

Micro Types (or Tiny Types)

- Further wrapping already-expressive types with even more expressive types
- Adds contextual clarity to reduce errors

```
public class OvertimeCalculator
{
  public OvertimeHours Calculate (HoursWorked worked,
        ContractedHours contracted)
  {
    var overtimeHours = worked.Hours - contracted.Hours;

    return new OvertimeHours(overtimeHours);
  }
}
```

Collection Aversion

- Instead of a collection of phone numbers,

  Use Expressive value objects of phone numbers

```
public class PhoneBook : ValueObject<PhoneBook> {

  public readonly PhoneNumber HomeNumber;
  public readonly PhoneNumber MobileNumber;
  public readonly PhoneNumber WorkNumber;

  public PhoneBook(PhoneNumber homeNum, PhoneNumber
        mobileNum, PhoneNumber workNum)   {

    this.HomeNumber = homeNum;
    this.MobileNumber = mobileNum;
    this.WorkNumber = workNum;
  }

}
```

Don’t forget …

ToString()

```
public override string ToString() {
    return _value;
}
```

Conversion

```
public static implicit operator string (PostCode postCode) {
    return postCode.Value;
}

public static explicit operator PostCode(string value) {
    return new PostCode(value);
}
```

Resousrces

- http://www.wrox.com/WileyCDA/WroxTitle/Patterns-Principles-and-Practices-of-Domain-Driven-Design.productCd-1118714709.html
- http://grabbagoft.blogspot.dk/2007/12/dealing-with-primitive-obsession.html
- http://www.refactoring.com/catalog/replaceDataValueWithObject.html
- http://fsharpforfunandprofit.com/posts/designing-with-types-single-case-dus/

Questions?

Thanks

- Speaker
  andrew.yh.chaa@gmail.com
  www.andrewchaa.me.uk
  @andrewchaa