Validating Input with FluentValidation



Vladimir Khorikov

@vkhorikov www.enterprisecraftsmanship.com



Recap: Validating Simple Properties



Validated simple properties

```
public class RegisterRequestValidator : AbstractValidator<RegisterRequest> {
    public RegisterRequestValidator() {
        RuleFor(x => x.Name).NotEmpty().Length(0, 200);
        RuleFor(x => x.Email).NotEmpty().Length(0, 150).EmailAddress();
        RuleFor(x => x.Address).NotNull().Length(0, 150);
    }
}
```

Fluent Interface pattern ——— Fluent Validation



Recap: Validating Simple Properties

```
public class RegisterRequestValidator : AbstractValidator<RegisterRequest> {
    public RegisterRequestValidator() {
        RuleFor(x => x.Name).NotEmpty().Length(0, 200);
        RuleFor(x => x.Email).NotEmpty().Length(0, 150).EmailAddress();
        RuleFor(x => x.Address).NotNull().Length(0, 150);
    }
}
```



Validates request data, not the domain class



Domain classes Z Data contracts





by Vladimir Khorikov

Building bullet-proof business line applications is a complex task. This course will teach you an in-depth guideline into refactoring from Anemic Domain Model into a rich, highly encapsulated one.

Resume Course

☐ Bookmark

((o)) Add to Channel

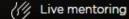


Table of contents Description Transcript Exercise files Discussion Learning Check Recommended

Expand all

Course Overview

✓ 🎵 1m 31s 🗸

Introduction

✓ 🖟 22m 24s 🗸

Introducing an Anemic Domain Model

Decoupling the Domain Model from Data Contracts

18m 31s

~

29m 46s

Course author



Vladimir Khorikov

Vladimir Khorikov is a Microsoft MVP and has been professionally involved in software development for more than 10 years.

Course info

	Intermediate
Rating	***** (73)
My rating	****
Duration	3h 36m
Released	13 Nov 2017

Share course

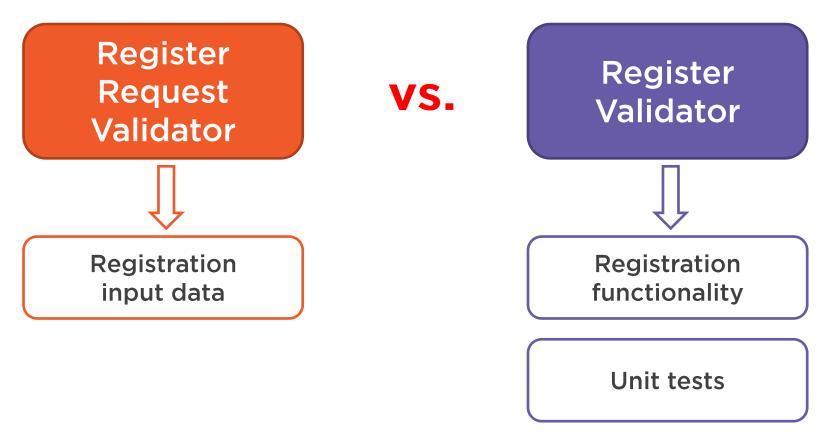








Recap: Validating Simple Properties



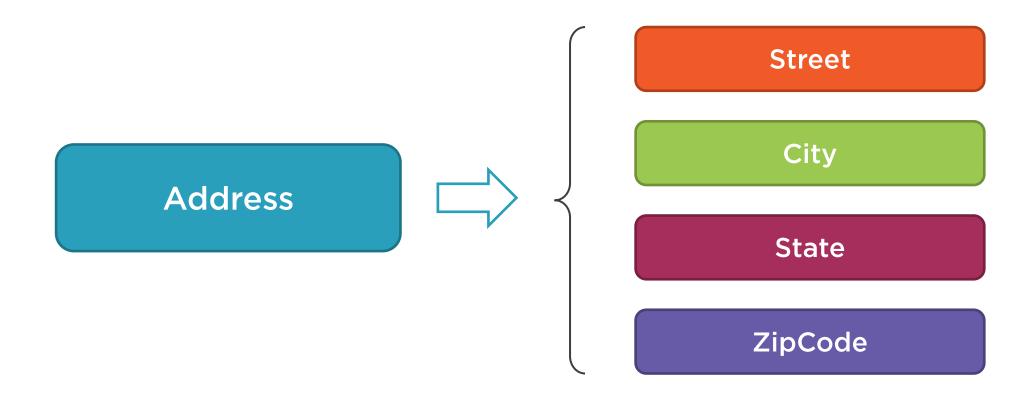








Validating Complex Properties





Validating Complex Properties



Why not use the same Address class?



Always keep the domain model separate from data contracts

Domain classes



Modeling the problem domain

Data contract



Public interface



Validating Complex Properties

Inlining validation rules

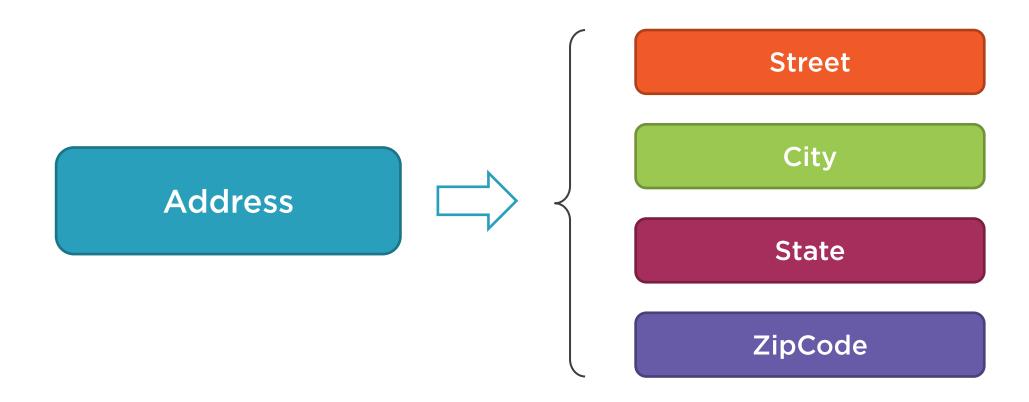
```
RuleFor(x => x.Address).NotNull();
RuleFor(x => x.Address.Street).NotEmpty().Length(0, 100).When(x => x.Address != null);
RuleFor(x => x.Address.City).NotEmpty().Length(0, 40).When(x => x.Address != null);
RuleFor(x => x.Address.State).NotEmpty().Length(0, 2).When(x => x.Address != null);
RuleFor(x => x.Address.ZipCode).NotEmpty().Length(0, 5).When(x => x.Address != null);
```







Recap: Validating Complex Properties





Recap: Validating Complex Properties

Inline nested rules

```
RuleFor(x => x.Address).NotNull();
RuleFor(x => x.Address.Street).NotEmpty().Length(0, 100).When(x => x.Address != null);
RuleFor(x => x.Address.City).NotEmpty().Length(0, 40).When(x => x.Address != null);
RuleFor(x => x.Address.State).NotEmpty().Length(0, 2).When(x => x.Address != null);
RuleFor(x => x.Address.ZipCode).NotEmpty().Length(0, 5).When(x => x.Address != null);
```





Separate validator

```
RuleFor(x => x.Address).NotNull().SetValidator(new AddressValidator());
```





Demo

Validating collections



Recap: Validating Collections



Validated the collection itself



Validated collection items



Extracted the rule into a separate validator



Recap: Validating Collections

```
public RegisterRequestValidator()
    RuleFor(x => x.Name).NotEmpty().Length(0, 200);
    RuleFor(x => x.Email).NotEmpty().Length(0, 150).EmailAddress();
    RuleFor(x => x.Addresses).NotNull().SetValidator(new AddressesValidator());
                                               Still duplicated
public EditPersonalInfoRequestValidator()
    RuleFor(x => x.Name).NotEmpty().Length(0, 200);
    RuleFor(x => x.Addresses).NotNull().SetValidator(new AddressesValidator());
```



Put this validation explicitly in each validator





Inheritance validation

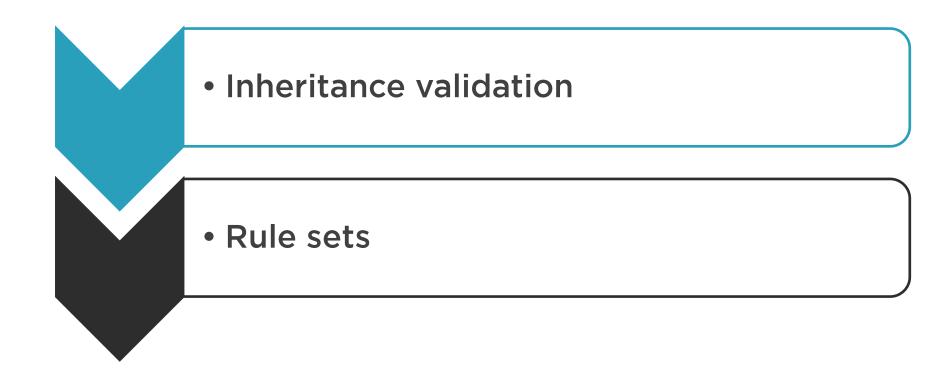




Rule sets



Recap: Inheritance Validation and Rule Sets





Recap: Inheritance Validation

```
public abstract class PhoneNumberDto {
    public string Number { get; set; }
}
public class USPhoneNumberDto : PhoneNumberDto {}
public class InternationalPhoneNumberDto : PhoneNumberDto {}

RuleFor(x => x.Phone).SetInheritanceValidator(x => {
        x.Add<USPhoneNumberDto>(new USPhoneNumberValidator());
        x.Add<InternationalPhoneNumberDto>(new InternationalPhoneNumberValidator());
});
```



Setting up rules polymorphically



Only applicable to domain classes



Recap: Rule Sets

```
public class StudentValidator : AbstractValidator<StudentDto> {
    public StudentValidator()
        RuleSet("Register", () =>
            RuleFor(x => x.Email).NotEmpty().Length(0, 150).EmailAddress();
        });
        RuleSet("EditPersonalInfo", () =>
            // No separate rules for EditPersonalInfo API yet
        RuleFor(x => x.Name).NotEmpty().Length(0, 200);
        RuleFor(x => x.Addresses).NotNull().SetValidator(new AddressesValidator());
```

Default rule set



Recap: Rule Sets

```
ValidationResult result = validator.Validate(request);

ValidationResult result = validator.Validate(request,
    options => options
    .IncludeRuleSets("Register"));

"Default" rule set
    "Register" rule set
```

ValidationResult result = validator.Validate(request,

.IncludeRuleSets("Register")

.IncludeRulesNotInRuleSet());

options => options



"Default" and "Register" rule sets

Recap: Rule Sets

```
public class StudentValidator : AbstractValidator<StudentDto> {
    public StudentValidator()
        RuleSet("Register", () =>
            RuleFor(x => x.Email).NotEmpty().Length(0, 150).EmailAddress();
        });
        RuleSet("EditPersonalInfo", () =>
        {
            // No separate rules for EditPersonalInfo API yet
        });
        RuleFor(x => x.Name).NotEmpty().Length(0, 200);
        RuleFor(x => x.Addresses).NotNull().SetValidator(new AddressesValidator());
```



Reusing validators



Don't reuse data contracts



Demo

Throwing exceptions



Throwing Exceptions



Don't use exceptions for validation

Validations =



Exceptional situation



Summary



Using the FluentValidation library

- Validation rules are defined in separate classes
- Rules are described fluently using the RuleFor method

Validating complex properties

Validating collections of objects

Used the ForEach method

Inheritance validation

Rule sets allow for combining of validation rules

Throwing exceptions



In the Next Module

Diving Deeper into FluentValidation

