Putting It All Together

Vladimir Khorikov

PROGRAMMER

@vkhorikov www.enterprisecraftsmanship.com

Overview

Applying immutability

Refactoring away from exceptions

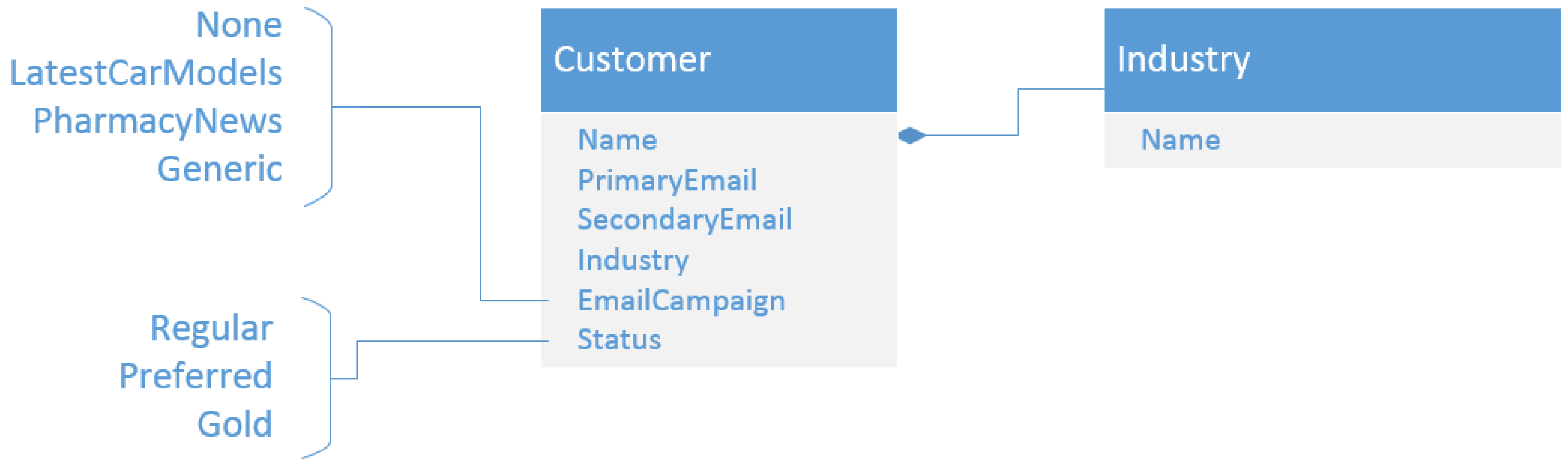
Getting rid of primitive obsession

Deal with nulls

Applying Railway-oriented programming

Working with a database

Domain Model Introduction



Industry	⇔	**Email Campaign**
Cars		**Latest cars models**
Pharmacy		**Pharmacy news**

Operations

- Create a customer

- Promote a customer
- Notification email should be sent out

- Disable emailing

- Change the industry
- Email campaign should be changed with it

- Get information about a customer

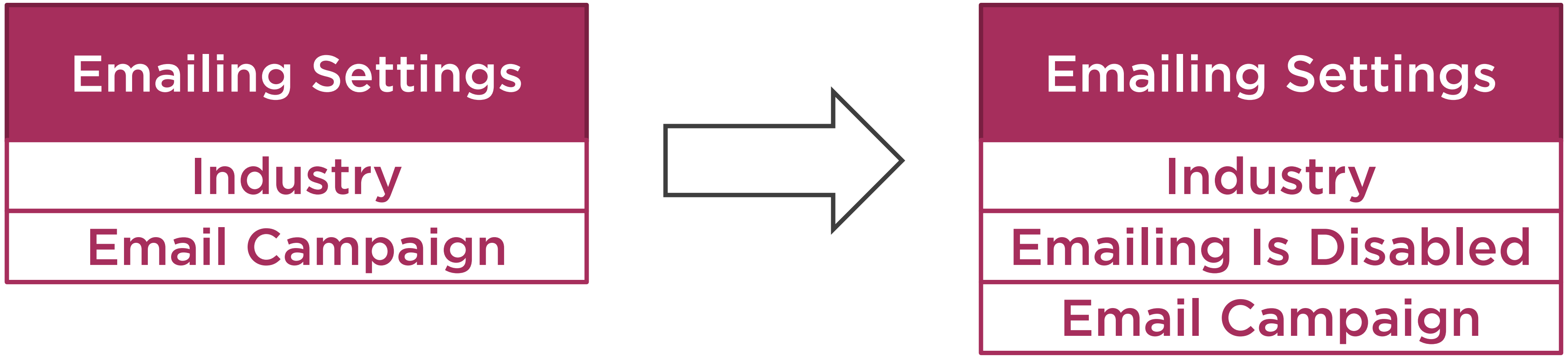
Recap: Refactoring Away from Exceptions

- **Stopped using exceptions to control the program flow**
- **Single generic exception handler to log all unexpected exceptions**
- **Caught known exceptions from the SMTP client on the lowest level possible**

Recap: Refactoring Away from Primitive Obsession

- Replaced strings with Value Objects
- Converted primitives into Value Objects on the boundary of the domain model
- Converted them back into primitives when they left the domain model

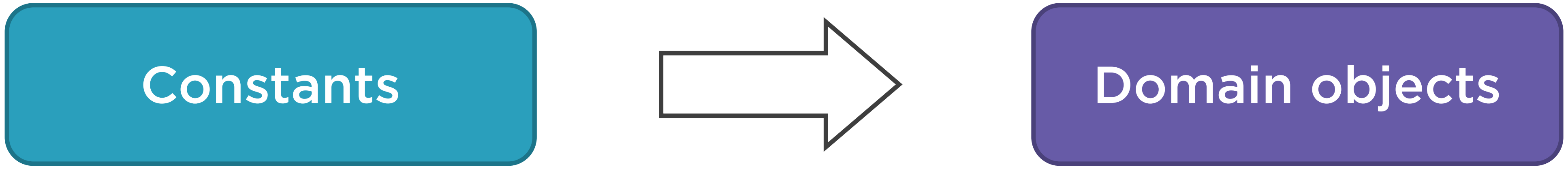
Recap: Refactoring to More Explicit Code



Recap: Making Nulls Explicit

- **NullGuard nuget library to check for nulls automatically**
- **Maybe type to explicitly mark nullable reference types**
- **Incoming nulls get converted into Maybe on the boundaries of the domain model**
- **They are converted back when they leave that boundary**

Recap: Representing Reference Data as Code



✓ Works with reference data only

✓ Cover with integration tests

http://bit.ly/1IZEwuy

Recap: Representing Reference Data as Code

```
Result<T>  ≠  Maybe<T>
```

```
Maybe<Customer>
~~Result<Customer>~~ GetById(int id)
```

```
Result<Maybe<Email>>
```

Recap: Railway-oriented Programming

```csharp
[HttpPost]
[Route("customers/{id}/promotion")]
public HttpResponseMessage Promote(long id)
{
    Maybe<Customer> customerOrNothing = _customerRepository.GetById(id);
    if (customerOrNothing.HasNoValue)
        return Error("Customer with such Id is not found: " + id);

    Customer customer = customerOrNothing.Value;

    if (!customer.CanBePromoted())
        return Error("The customer has the highest status possible");

    customer.Promote();

    Result result = _emailGateway.SendPromotionNotification(customer.PrimaryEmail, customer.Status);
    if (result.IsFailure)
        return Error(result.Error);

    return Ok();
}
```

Recap: Railway-oriented Programming

```
[HttpPost]
[Route("customers/{id}/promotion")]
public HttpResponseMessage Promote(long id)
{
    return _customerRepository.GetById(id)
        .ToResult("Customer with such Id is not found: " + id)
        .Ensure(customer => customer.CanBePromoted(), "The customer has the highest status possible")
        .OnSuccess(customer => customer.Promote())
        .OnSuccess(customer => _emailGateway.SendNotification(customer.PrimaryEmail, customer.Status))
        .OnBoth(result => result.IsSuccess ? Ok() : Error(result.Error));
}
```

✓ **OnSuccess = Bind**

✓ **Result = Monad**

Module Summary

Refactoring away from exceptions

Avoiding primitive obsession

Converting primitives into value objects

Making implicit assumptions explicit

Disallowing nullable reference types by default

Representing reference data as code

Using the railway-oriented programming approach

Resource List

Source code	https://github.com/vkhorikov/FuntionalPrinciplesCsharp
	http://bit.ly/1U3bkcz
C#: Non-nullable Reference Types	http://blog.coverity.com/2013/11/20/c-non-nullable-reference-types/
	http://bit.ly/1TW4ofH
Proposal: Nullable reference types and nullability checking	https://github.com/dotnet/roslyn/issues/5032
	http://bit.ly/1VTxlIi
Railway-oriented programming approach	https://vimeo.com/113707214
Database versioning best practices	http://enterprisecraftsmanship.com/2015/08/10/database-versioning-best-practices/
	http://bit.ly/1IZEwuy
Fail Fast principle	http://enterprisecraftsmanship.com/2015/09/15/fail-fast-principle/
	http://bit.ly/1RrHvj8

Course Summary

Principles that lie at the foundation of functional programming

- Method signature honesty
- Referential transparency

Side effects and exceptions make your code dishonest about the outcome it may produce

Primitive obsession makes your code dishonest about its input parts

Nulls make your code dishonest about both its inputs and outputs

Contacts

vladimir.khorikov@gmail.com

@vkhorikov

http://enterprisecraftsmanship.com/