Avoiding Primitive Obsession

Vladimir Khorikov
PROGRAMMER

@vkhorikov www.enterprisecraftsmanship.com

Primitive obsession stands for using primitive types for domain modeling.

Drawbacks of Primitive Obsession

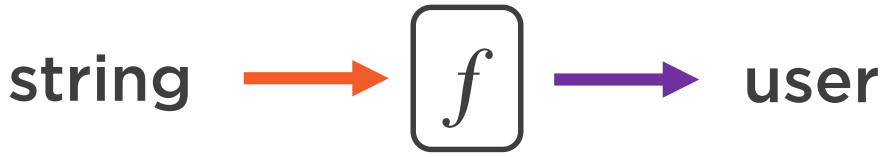
```
public class User
{
    public string Email { get; }

    public User(string email)
    {
        Email = email;
    }
}
```

Drawbacks of Primitive Obsession

```
public class User
{
    public string Email { get; }

    public User(string email)
    {
        if (string.IsNullOrWhiteSpace(email))
            throw new ArgumentException("Email should not be empty");

        email = email.Trim();
        if (email.Length > 256)
            throw new ArgumentException("Email is too long");

        if (!email.Contains("@"))
            throw new ArgumentException("Email is invalid");

        Email = email;
    }
}
```

Drawbacks of Primitive Obsession

```
public class UserFactory
{
    public User CreateUser(string email)
    {
        return new User(email);
    }
}
```



Dishonest signature

```
public int Divide(int x, int y)
{
    return x / y;
}
```

Dishonest signature

Drawbacks of Primitive Obsession

```
public class Organization
{
    public string PrimaryEmail { get; }

    public Organization(string primaryEmail)
    {
        PrimaryEmail = primaryEmail;
    }
}
```

Drawbacks of Primitive Obsession

```
public class Organization
{
    public string PrimaryEmail { get; }

    public Organization(string primaryEmail)
    {
        if (string.IsNullOrWhiteSpace(primaryEmail))
            throw new ArgumentException("Email should not be empty");

        primaryEmail = primaryEmail.Trim();
        if (primaryEmail.Length > 256)
            throw new ArgumentException("Email is too long");

        if (!primaryEmail.Contains("@"))
            throw new ArgumentException("Email is invalid");

        PrimaryEmail = primaryEmail;
    }
}
```

Drawbacks of Primitive Obsession

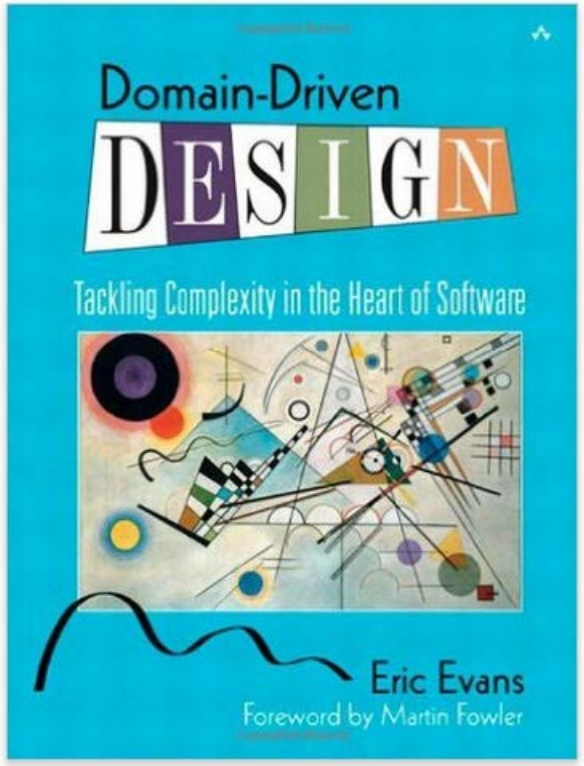
```
public class Organization
{
    public string PrimaryEmail { get; }
    public string SecondaryEmail { get; }

    public Organization(string primaryEmail, string secondaryEmail)
    {
        Validate(primaryEmail, secondaryEmail);

        PrimaryEmail = primaryEmail;
        SecondaryEmail = secondaryEmail;
    }

    private void Validate(params string[] emails)
    {
        /* Perform the validation here */
    }
}
```

Drawbacks of Primitive Obsession

Makes code dishonest

Violates the DRY principle

How to Get Rid of Primitive Obsession



Domain-Driven Design: Tackling Complexity in the Heart of Software

By Eric Evans

Domain-Driven Design in Practice

by Vladimir Khorikov

A descriptive, in-depth walk-through for applying Domain-Driven Design principles in practice.

▶ Resume Course

Table of contents | Description | Transcript | Exercise files | Discussion | Learning Check | Recommended

Introduction 29m 31

Starting with the First Bounded Context 46m 18

Introducing UI and Persistence Layers ✓ 33m 20

Starting with Snack Machine

Recap: Starting with Snack Machine

Entities vs. Value Objects

How to Recognize a Value Object in Your Domain Model?

Entity Base Class

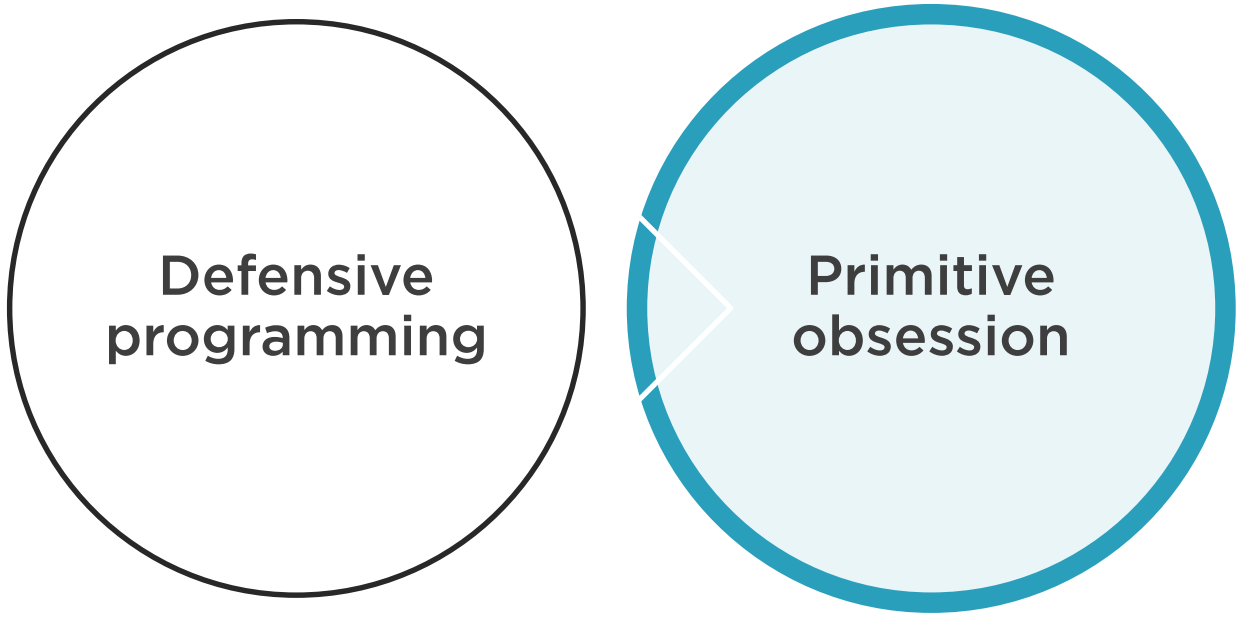
Demo: Entity Base Class

Recap: Entity Base Class

Value Object Base Class

Recap: Value Object Base Class

Primitive Obsession and Defensive Programming



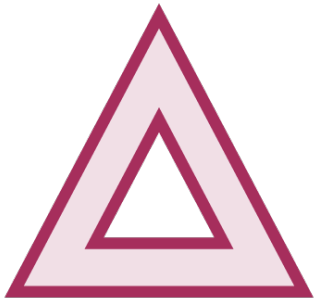
Primitive Obsession and Defensive Programming

```
public void ProcessUser(string name) {
    if (string.IsNullOrWhiteSpace(name)) throw new ArgumentException(nameof(name));
    if (name.Trim().Length > 100) throw new ArgumentException(nameof(name));

    /* Processing code */
}

public void CreateUser(string name) {
    if (string.IsNullOrWhiteSpace(name)) throw new ArgumentException(nameof(name));
    if (name.Trim().Length > 100) throw new ArgumentException(nameof(name));

    /* Creation code */
}

public void UpdateUser(string name) {
    if (string.IsNullOrWhiteSpace(name)) throw new ArgumentException(nameof(name));
    if (name.Trim().Length > 100) throw new ArgumentException(nameof(name));

    /* Update code */
}
```

Primitive Obsession and Defensive Programming

```
public void ProcessUser(UserName name) {
    if (name != null)
        throw new ArgumentNullException(nameof(name));

    /* Processing code */
}

public void CreateUser(UserName name) {
    if (name != null)
        throw new ArgumentNullException(nameof(name));

    /* Creation code */
}

public void UpdateUser(UserName name) {
    if (name != null)
        throw new ArgumentNullException(nameof(name));

    /* Update code */
}
```

Primitive Obsession: Limitations



Don’t create types for all domain concepts

Primitive Obsession: Limitations

```
                    decimal
moneyAmount : MoneyAmount
```

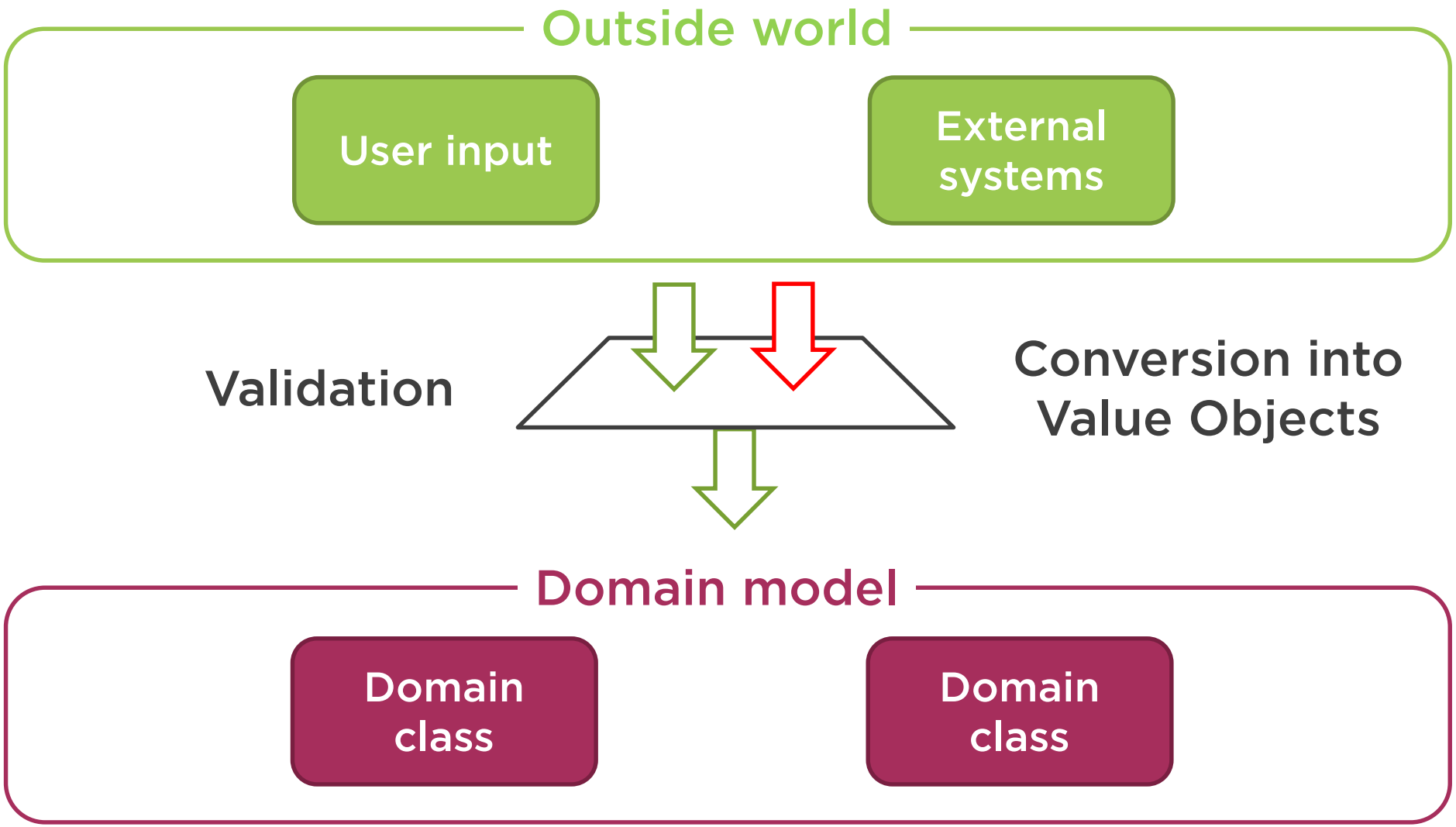
Primitive Obsession: Limitations

`Email`

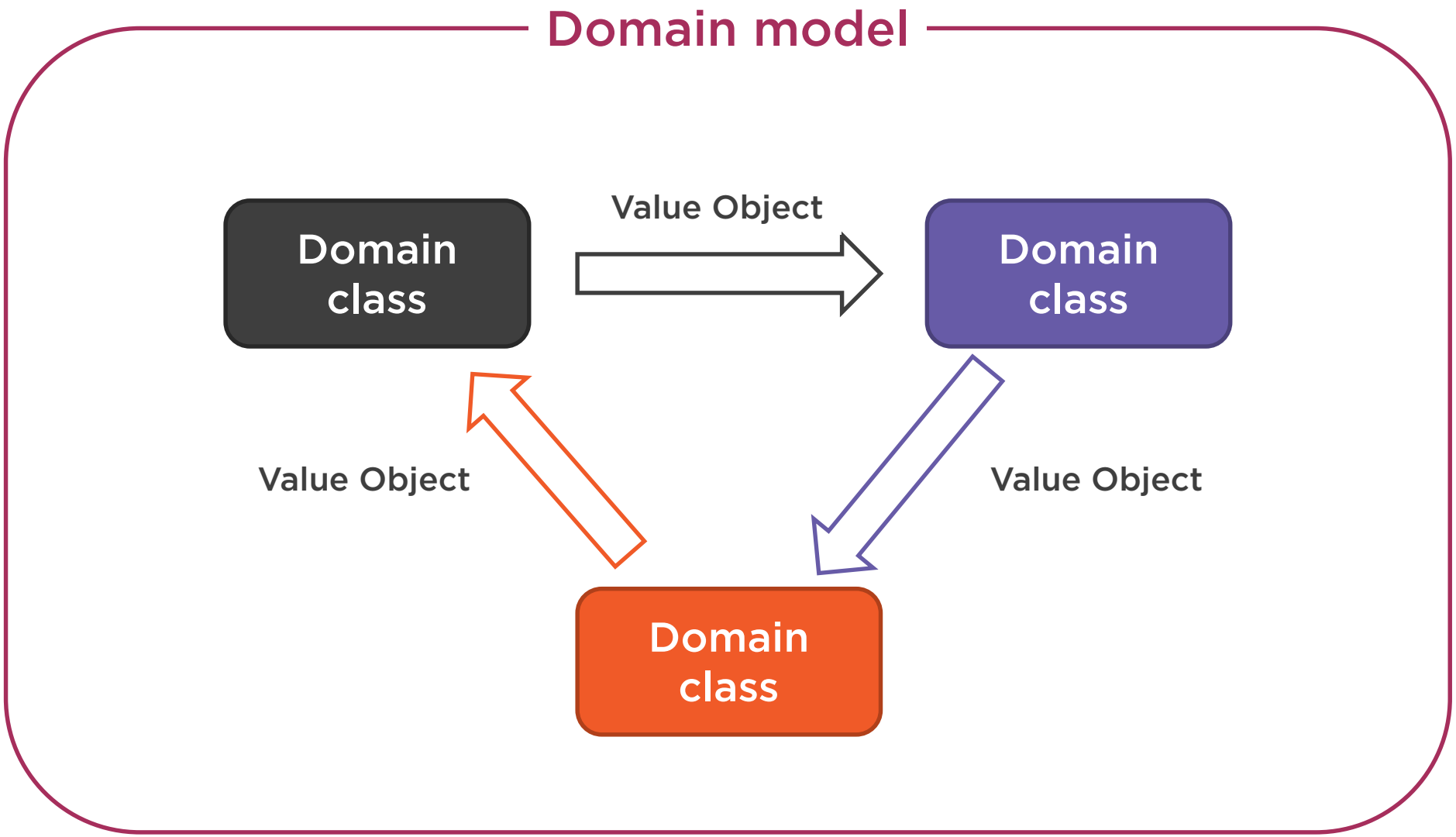
`UserName`

`MoneyAmount`

Where to Convert Primitive Types into Value Objects?



Where to Convert Primitive Types into Value Objects?



Where to Convert Primitive Types into Value Objects?

```
public void Process(string oldEmail, string newEmail)
{
    Result<Email> oldEmailResult = Email.Create(oldEmail);
    Result<Email> newEmailResult = Email.Create(newEmail);

    if (oldEmailResult.IsFailure || newEmailResult.IsFailure)
        return;

    string oldEmailValue = oldEmailResult.Value;
    Customer customer = GetCustomerByEmail(oldEmailValue);
    customer.Email = newEmailResult.Value;
}
```

```
public void Process(Email oldEmail, Email newEmail)
{
    Customer customer = GetCustomerByEmail(oldEmail);
    customer.Email = newEmail;
}
```

Recap: Refactoring Away from Primitive Obsession

- Removed validation logic duplications
- Created a single authoritative source of the domain knowledge
- Method signature honesty
  Stronger type system
- No need to validate values passed in

Summary

Primitive obsession

- Makes your code dishonest
- Encourages code duplication

Create a separate class for each concept in your domain model

Don’t create classes for simple concepts

Always use Value Objects inside your domain model

Convert them into primitive types when they leave the domain model

Example of refactoring from primitive obsession

In the Next Module

How to work with nulls