

Elements of a Domain Model: Value Objects & Services



Steve Smith
Force Multiplier for
Dev Teams
[@ardalis](https://twitter.com/ardalis) ardalis.com



Julie Lerman
Software coach,
DDD Champion
[@julielerman](https://twitter.com/julielerman) thedatafarm.com

Module Overview



Get acquainted with value objects

How entities embrace value objects

Advice from Eric Evans and Vaughn Vernon

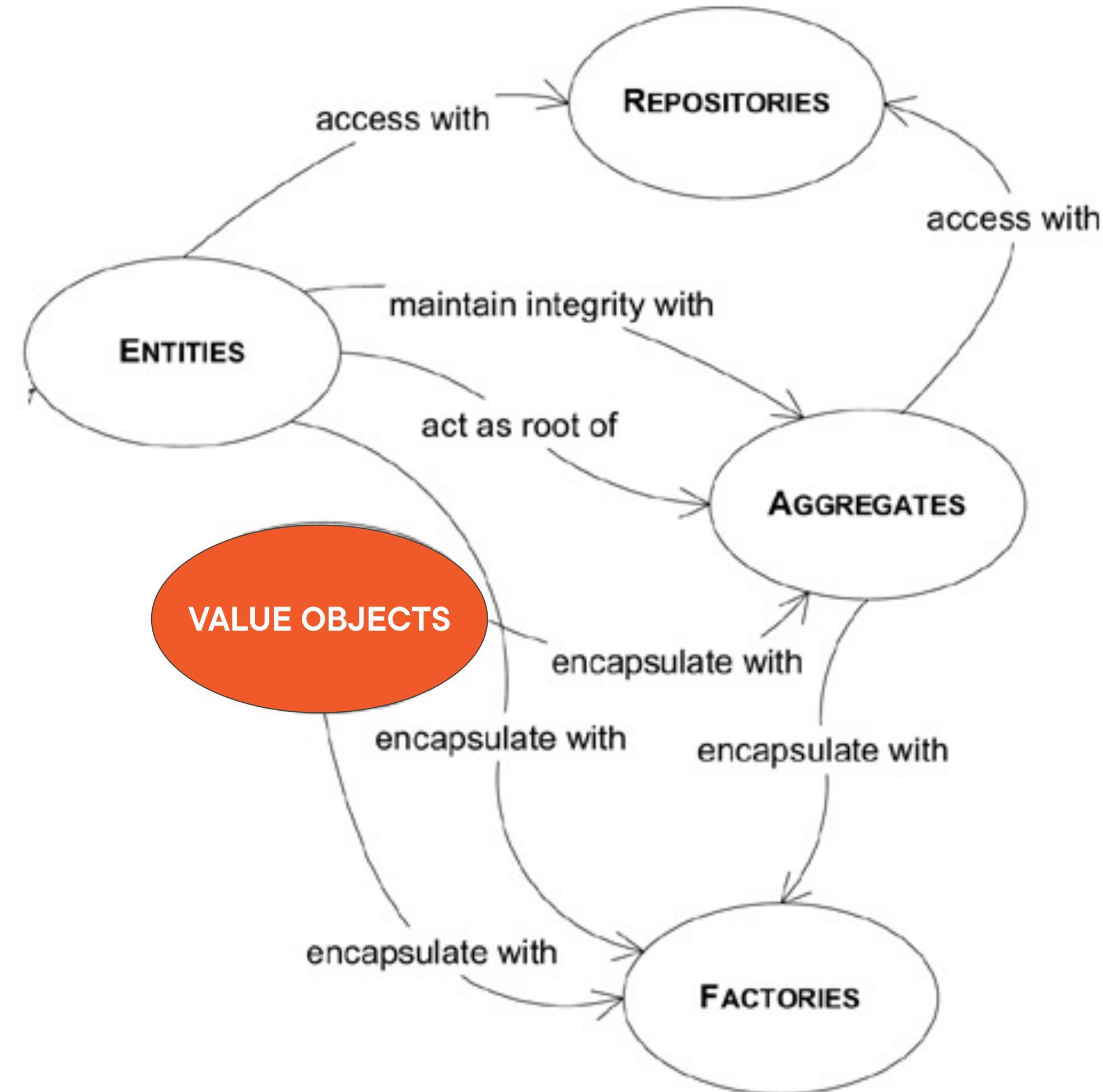
Implement value objects in code

What are domain services?

Features & examples of domain services

Getting Acquainted with Value Objects

Value Objects in DDD Mind Map



Value Object Attributes

Measures, quantifies, or describes a thing in the domain

Identity is based on composition of values

Immutable

Compared using all values

No side effects



Putting Value Objects into Context for .NET Devs

Value Types

Defined with structs

Reference Types

Defined with classes

DDD Entities &
Value Objects

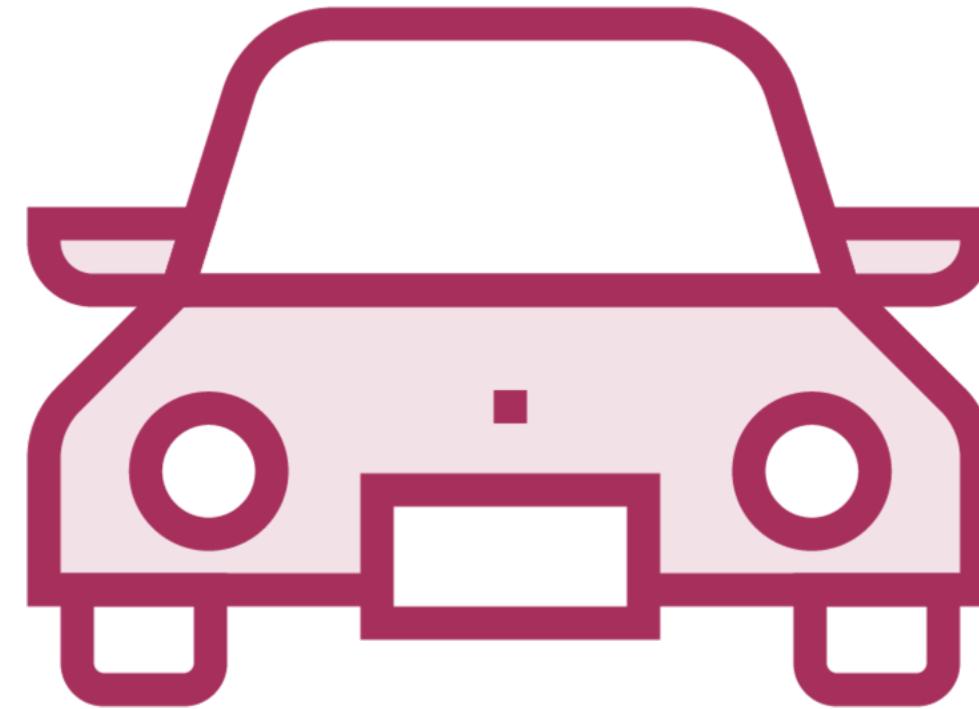
**Typically defined
with classes**

Recognizing Commonly Used Value Objects

String is a value object

String: Our Favorite Value Object

C A R



String: Our Favorite Value Object

C A T



Marlon lives with author, John Elliott

String: Our Favorite Value Object

S C A R



String: Our Favorite Value Object

S CAR



D O GDESS

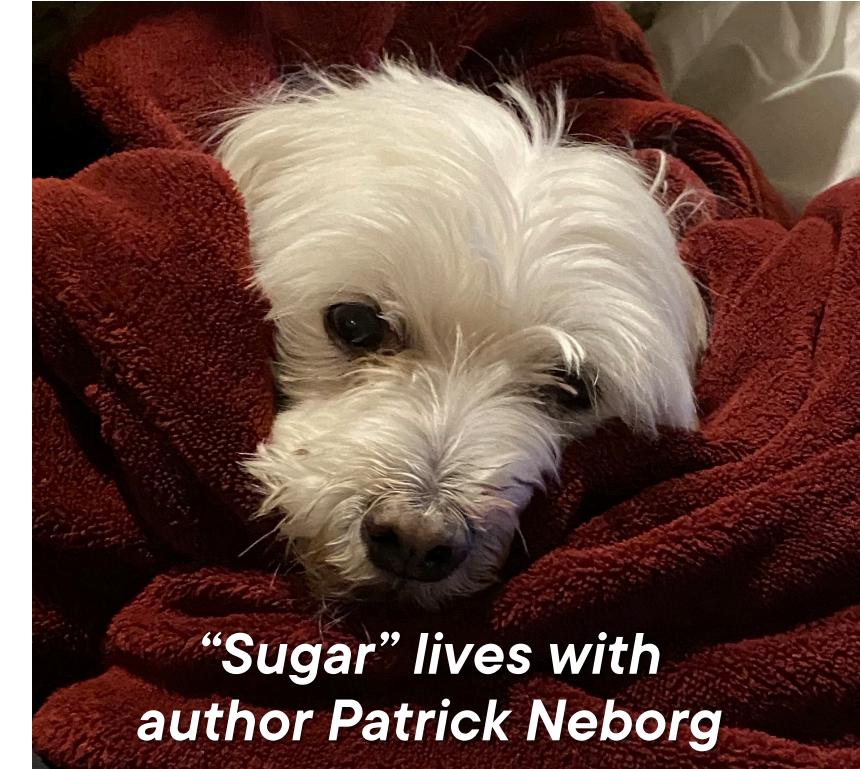


String: Our Favorite Value Object

S CAR



D O GDESS



*“Sugar” lives with
author Patrick Neborg*

String Methods Respect Immutability

Replace (StringA, StringB)

Returns a new string in which all occurrences StringA in the current instance are replaced with StringB.

ToUpper()

Returns a copy of this string converted to uppercase.

ToLower()

Returns a copy of this string converted to lowercase.

Money is a Great Value Object

Company Worth \$ 50,000,000

Money is a Great Value Object

Company Worth	\$US	50,000,000
	\$CN	
	\$AU	

Money is a Great Value Object

Company Worth \$US 50,000,000



Source: <http://fit.c2.com/wiki.cgi?WholeValue> (Ward Cunningham)

Money is a Great Value Object

Company Worth **\$US** **50,000,000**

Company
- Id
- Worth Unit
- Worth Amount

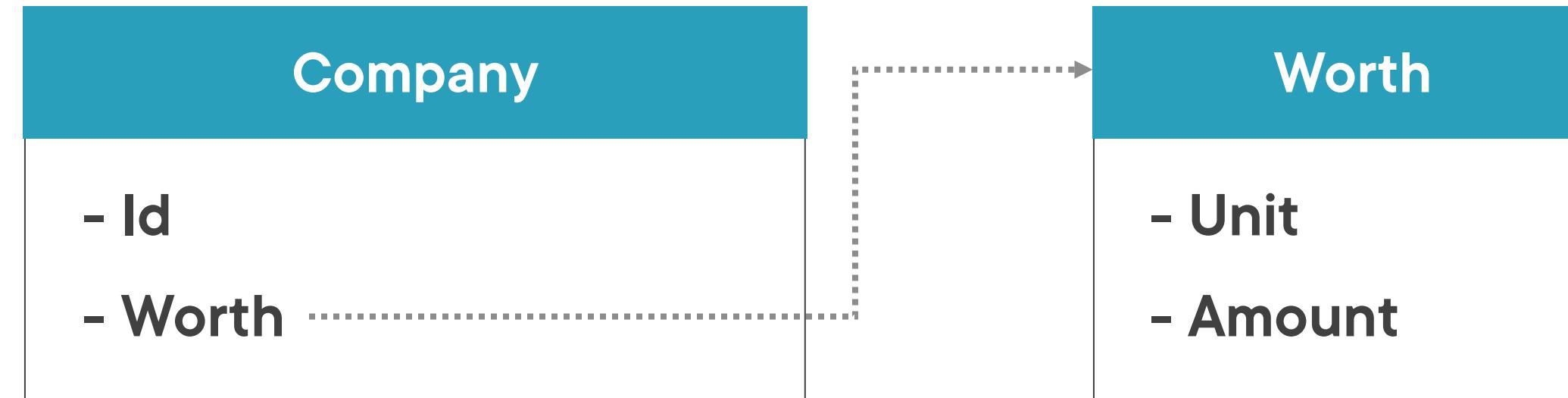
Money is a Great Value Object

Company Worth **\$US** **50,000,000**

Company
- Id
- Worth Amount

Money is a Great Value Object

Company Worth \$US 50,000,000



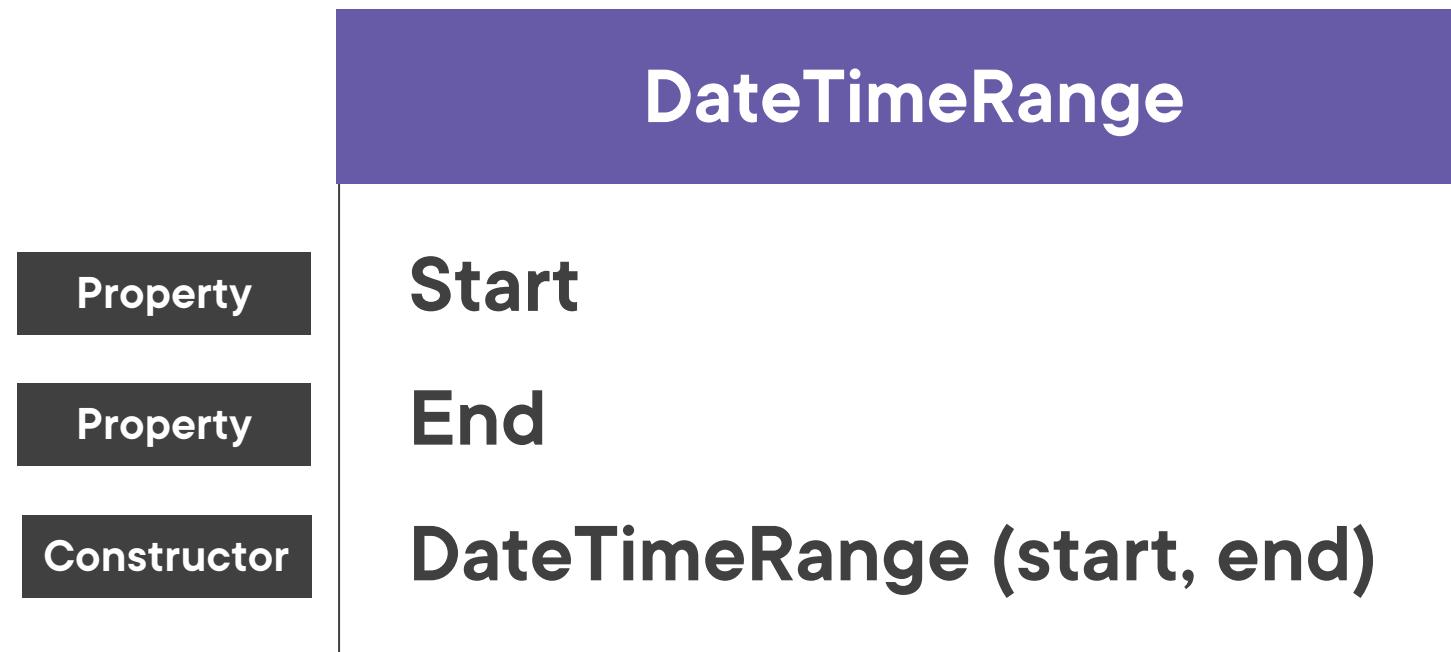
`DateTimeRange` as Value Object

Patient Appointment

10:00 am Jan 4 – 11:00 am Jan 4

Staff Meeting

2:00 pm Feb 1 – 3:15 pm Feb 1



`DateTimeRange` as Value Object

Patient Appointment

10:00 am Jan 4 – 11:00 am Jan 4

Staff Meeting

2:00 pm Feb 1 – 3:15 pm Feb 1

<code>DateTimeRange</code>	<code>Appointment</code>	<code>Meeting</code>
<code>Start</code>		
<code>End</code>		
<code>DateTimeRange (start, end)</code>	<ul style="list-style-type: none">- <code>ClientId</code>- <code>DoctorId</code>- <code>PatientId</code>- <code>DateTimeRange</code>	<ul style="list-style-type: none">- <code>RoomId</code>- <code>StaffAttending</code>- <code>DateTimeRange</code>

Getting More Insight from Eric Evans and Vaughn Vernon

It may surprise you to learn that we should strive to model using Value Objects instead of Entities wherever possible. Even when a domain concept must be modeled as an Entity, the Entity's design should be biased toward serving as a value container rather than a child Entity container.

Vaughn Vernon – Implementing Domain Driven Design

When Considering Domain Objects

Our Instinct:

- 1. Probably an entity**
- 2. Maybe a value object**

Vaughn Vernon's guidance:

- 1. Is this a value object?**
- 2. Otherwise, an entity**

Value Objects Can Be Used for Identifiers

ClientIdValueObject.cs

```
public class ClientId
{
    public readonly Guid Id;
    public ClientId()
    {
        Id = Guid.NewGuid();
    }
    public ClientId(Guid id)
    {
        Id = id;
    }
}

[Equality and Hash override code]
}
```

Explicit ID Value Objects Instead of Ints/GUIDs

Helps to Avoid Errors in Passed Parameters

Client.cs

```
public class Client : BaseEntity<ClientIdValueObject>
{
    . . .
}
```

SomeService.cs

```
public class SomeService
{
    public void CreateAppointmentFor(ClientIdValueObject clientId,
PatientIdValueObject patientId)
{
    . . .
}
}
```

Value Objects Can Be Used for Identifiers

Client.cs

```
public class Client
{
    public ClientIdValueObject Id {get; set;}
}

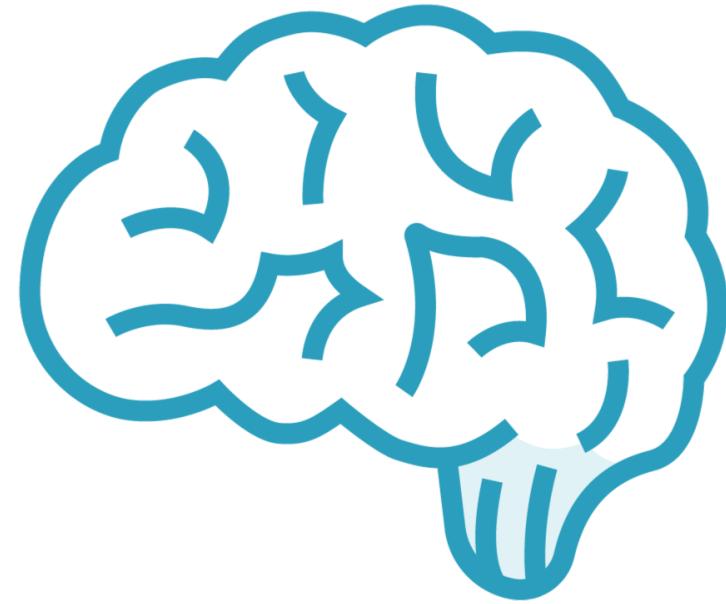
// or

public class Client : BaseEntity<ClientIdValueObject>
{
    // Id property provided by base type
}
```

“I think that value objects are a really good place to put methods and logic...because we can do our reasoning without side effects and identity, all those things that make logic tricky. We can put functions on those value objects and do the pure reasoning there.”

Eric Evans

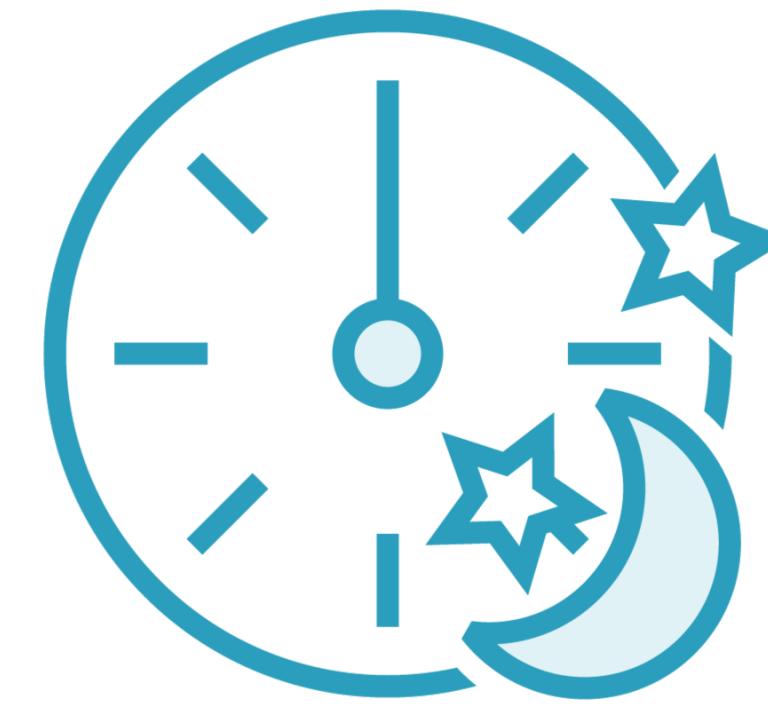
Date Libraries as Value Object



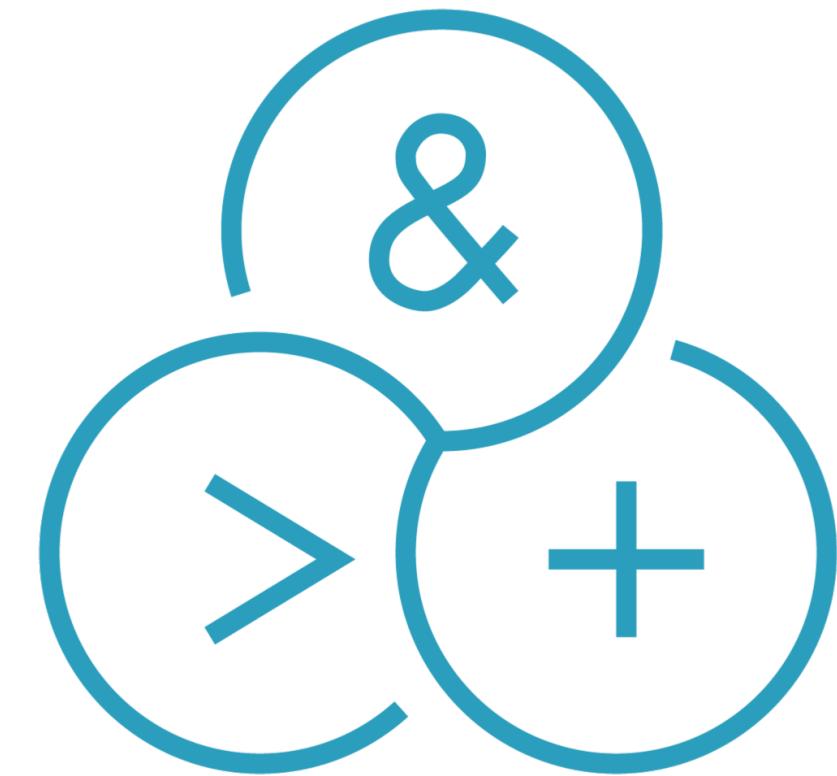
Calculate age



**Calculate
date ranges**



**Perform complex
time operations**



... and more

“Dates are a classic value object and
there’s all kinds of logic with them.”

Eric Evans

Implementing Value Objects in Code

Our DateTimeRange Value Object

```
public class DateTimeRange : ValueObject
{
    public DateTime Start { get; private set; }
    public DateTime End { get; private set; }

    public DateTimeRange(DateTime start, DateTime end)
    {
        // Ardalис.GuardClauses supports extensions with custom guards per project
        Guard.Against.OutOfRange(start, nameof(start), start, end);
        Start = start;
        End = end;
    }

    public DateTimeRange(DateTime start, TimeSpan duration)
        : this(start, start.Add(duration))
    {
    }

    // additional methods in next slide
}
```

The state of a value object
should not be changed once it
has been created.

Methods in Our DateTimeRange Value Object

```
public class DateTimeRange : ValueObject
{
    // properties and constructors

    public DateTimeRange NewEnd(DateTime newEnd)
    {
        return new DateTimeRange(this.Start, newEnd);
    }

    public bool Overlaps(DateTimeRange dateTimeRange)
    {
        return this.Start < dateTimeRange.End && this.End > dateTimeRange.Start;
    }

    // used by base ValueObject type to implement equality
    protected override IEnumerable<object> GetEqualityComponents()
    {
        yield return Start;
        yield return End;
    }
}
```

Our Animal Type Value Object

```
public class AnimalType : ValueObject
{
    public string Species { get; private set; }
    public string Breed { get; private set; }

    public AnimalType(string species, string breed)
    {
        Species = species;
        Breed = breed;
    }

    // used by base ValueObject type to implement equality
    protected override IEnumerable<object> GetEqualityComponents()
    {
        yield return Breed;
        yield return Species;
    }
}
```



**Can you share advice about moving logic out
of entities into value objects?**



**Generic logic
makes sense in
value objects**

**It's easier to test
logic that's in a
value object**

**Entity becomes an
orchestrator**

A higher level of abstraction in entities can lead you to a more precise ubiquitous language.

**Higher
level of
abstraction
in entities**



**More
precise
ubiquitous
language**

Working Towards a More Concise Language

**Extract
primitives
from an
entity**

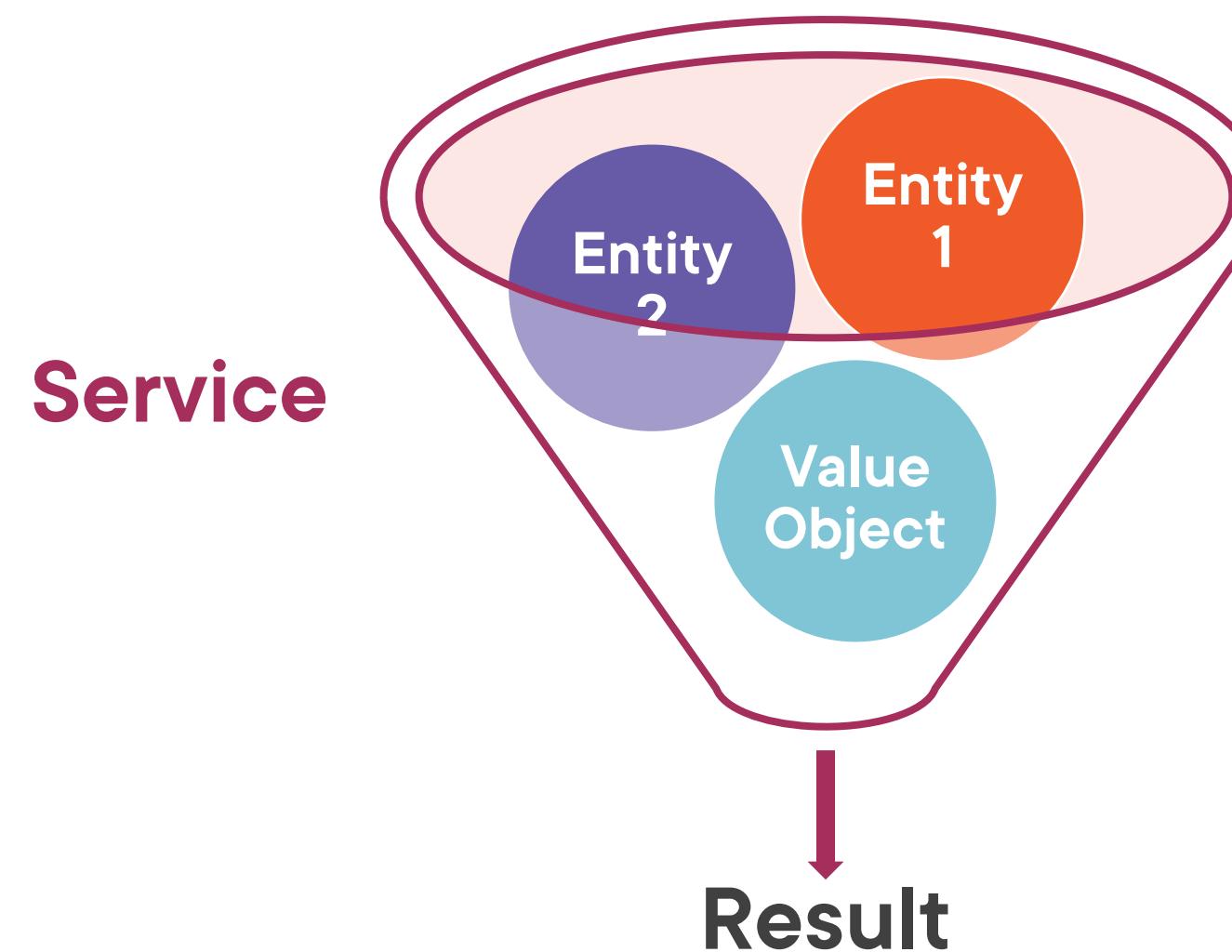
**into a value
object**



Understanding Domain Services

Some operations make more
sense in a *domain* service.

Domain Service Orchestrates Processes Across Objects



Features of a Domain Service

Not a natural part of an entity or value object

Has an interface defined in terms of other domain model elements

Stateless, but may have side effects

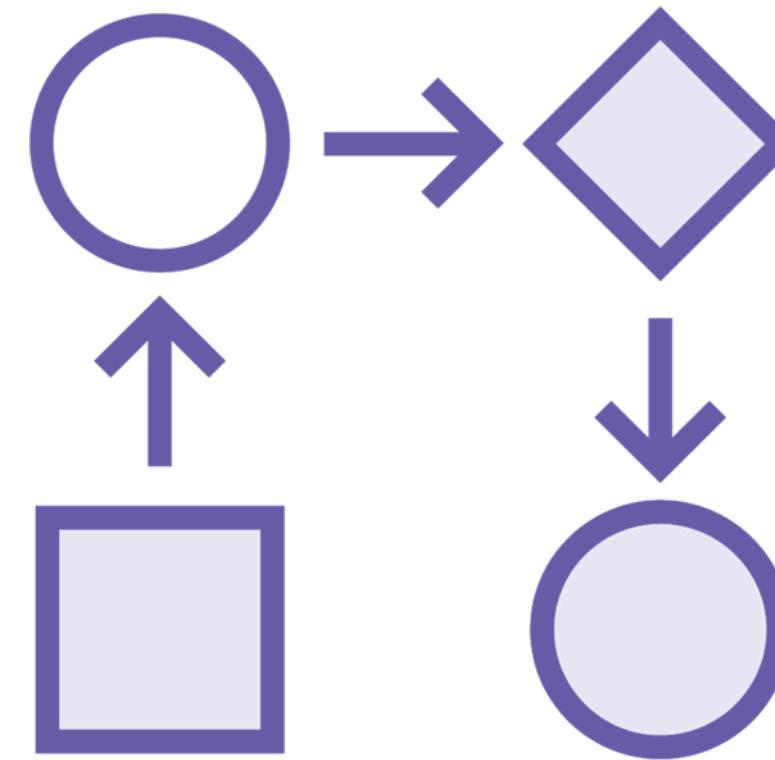
Lives in the core of the application

Examples of Services in Different Layers



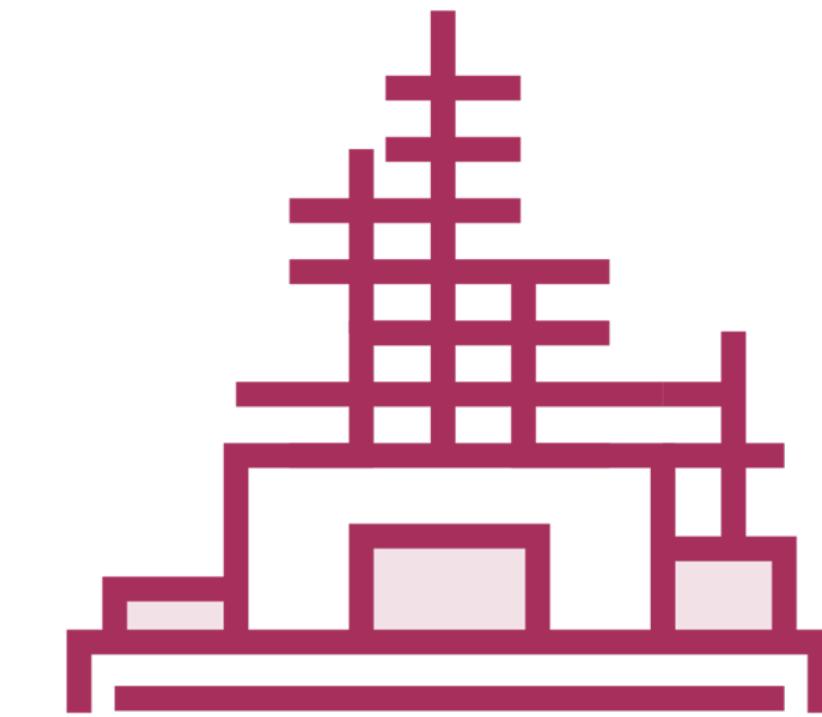
UI and App

Message Sending
Message Processing
XML Parsing
UI Services



Domain

Orchestrating workflow
Transfer Between
Accounts
Process Order



Infrastructure

Send Email
Log to a File

Module Review and Resources

Key Terms from this Module

Immutable

Refers to a type whose state cannot be changed once the object has been instantiated

Value Object

An immutable class whose identity is dependent on the combination of its values

Key Terms from this Module

Domain Services

Provide a place in the model to hold behavior that doesn't belong elsewhere in the domain

Side Effects

Changes in the state of the application or interaction with the outside world (e.g., infrastructure)

Key Takeaways



Value objects are used to measure quantify or describe.

They are used as a property of an entity.

They are identified by the composition of their values.

Value objects are immutable and should have no side effects.

Strings and dates are great examples of value objects.

Domain services orchestrate across different parts of the domain model.

Watch out for overuse of domain services!

Up Next:
Tackling Complexity with Aggregates



Thanks to Eric Evans for
his great advice and insights

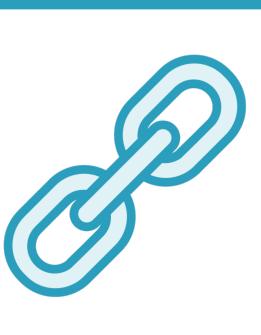
Resources Referenced in This Course



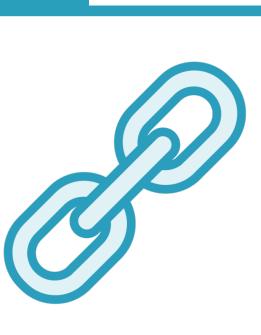
Domain-Driven Design in C#9: Immutable Value Objects
Julie Lerman on Pluralsight blog: bit.ly/PSBlogValueObjects



Support for Value Objects in C#, Steve Smith blog post:
ardalis.com/support-for-value-objects-in-csharp



Vaughn Vernon website: vaughnvernon.com



Eric Evans website: domainlanguage.com

Elements of a Domain Model: Value Objects & Services



Steve Smith
Force Multiplier for
Dev Teams
[@ardalis](https://twitter.com/ardalis) ardalis.com



Julie Lerman
Software coach,
DDD Champion
[@julielerman](https://twitter.com/julielerman) thedatafarm.com