Elements of a Domain Model

Organizing your sub domains using the key patterns of DDD

Julie Lerman
TheDataFarm.com
@julielerman

Steve Smith
Ardalis.com
@ardalis

pluralsight
hardcore developer training

In This Module

- Focus on Domain
- Focus on Behaviors
- Rich vs. Anemic Domain Models
- Entities
- Associations
- Value Objects
- Services


DDD
domain
BOUNDED
CONTEXTS
MODEL
CONVERSATION
APPLICATION
CUSTOMER
LANGUAGE
OBJECTS
BUILT
SOLUTION
TOUR
INCORRECT
using
corrections
projects
View
INCLUDE
NAMES
FLESH
MAPPED
ubiquitous
TECHNICAL
PIECES
THINKING
NOTES
MAKE
STUFF
IDS
fact
Reference
WORRY
schema
database
role
TALK
POINT
VERNON
ENTITIES
SOFTWARE
outside
FEATURE
CONTROL
MODULES
PACKAGES
MANAGING
ish
come
EF
DON
PROGRAMMERS

D

is for

DOMAIN

> [The Domain Layer is] responsible for representing concepts of the business, information about the business situation, and business rules. State that reflects the business situation is controlled and used here, even though the technical details of storing it are delegated to the infrastructure.*This layer is the heart of business software.*

— **Eric Evans**
Domain Driven Design

Behaviors

Schedule an appointment for a checkup
Note a pet's weight
Request lab work
Notify pet owner of vaccinations due
Accept a new patient
Book a room

Not Attributes

Appointment.Time **Pet.Name** **Owner.Telephone** **Room.Number**

Anemic

Domain Models

Rich

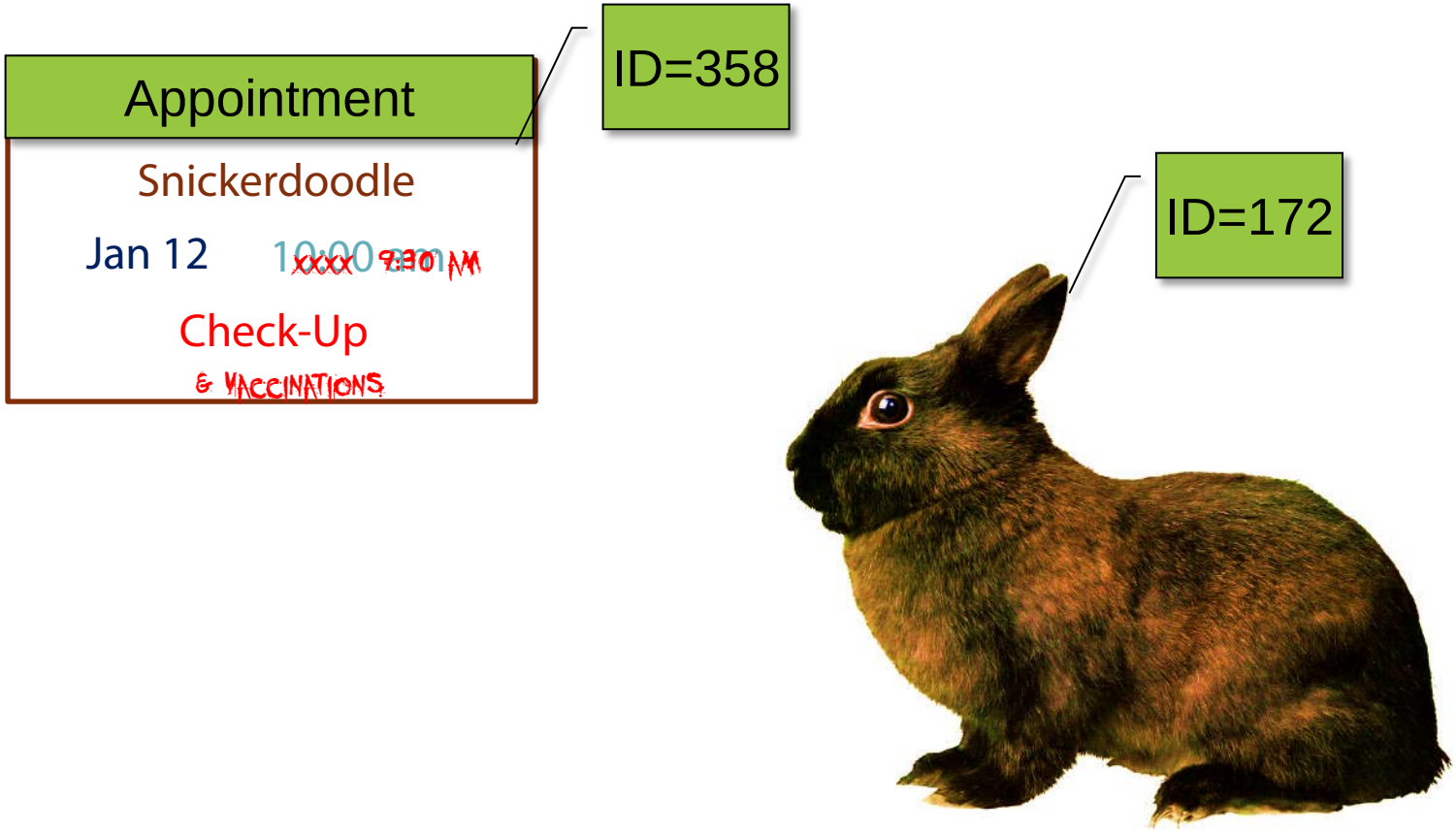
Domain Models

“Many objects are not fundamentally defined by their attributes, but rather by a **thread of continuity** and **identity**.”

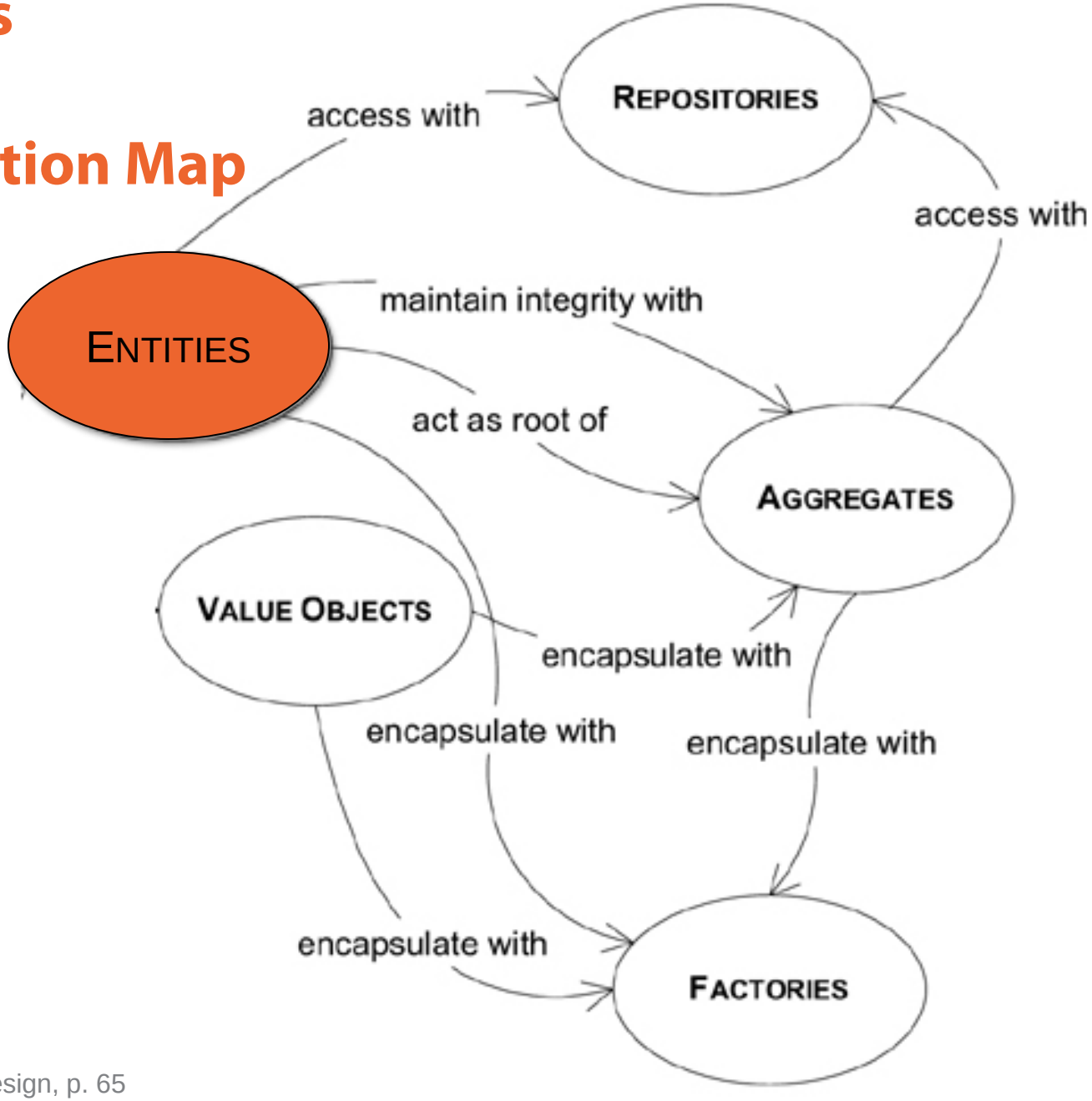
— Eric Evans
Domain-Driven Design

Entity

Entities Have Identity & Are Mutable

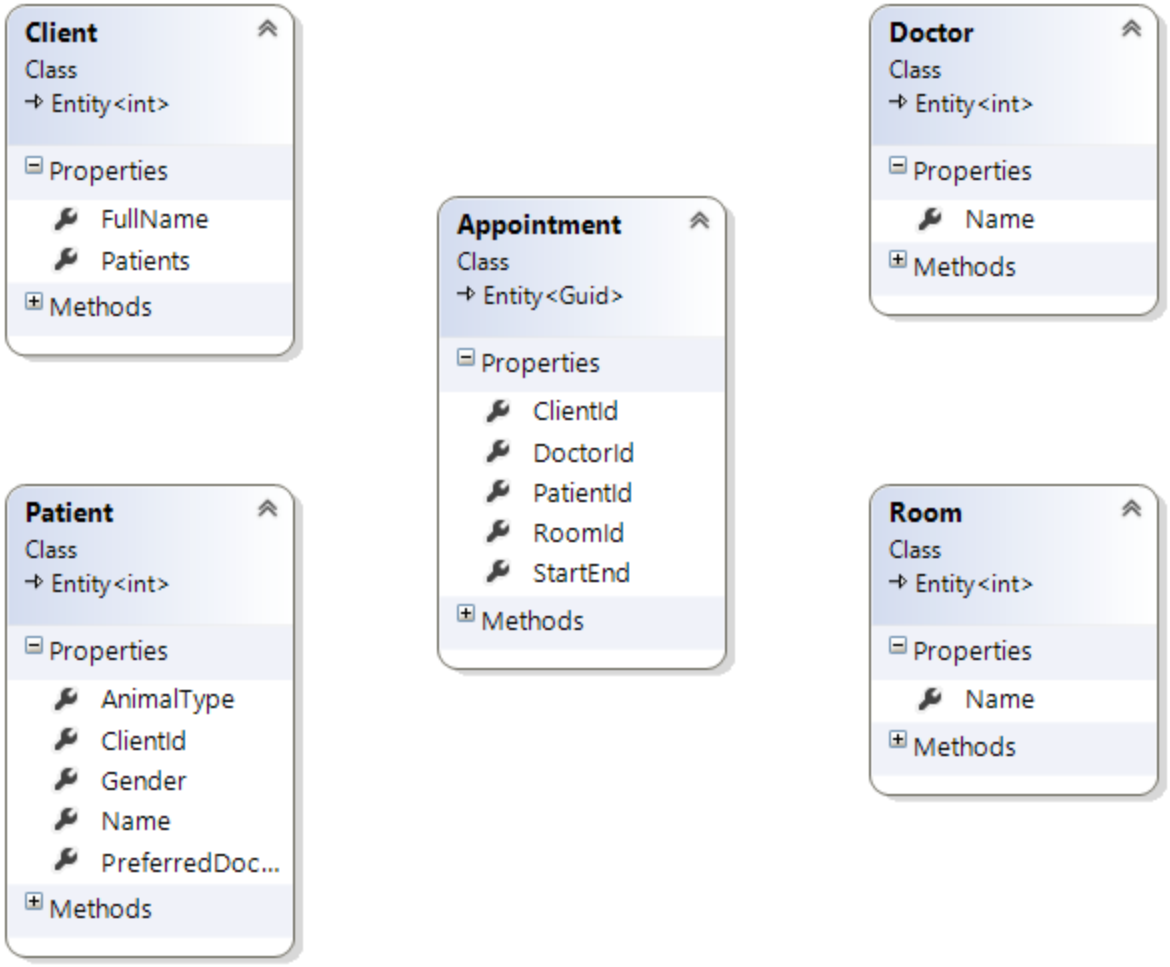


Entities in the Navigation Map



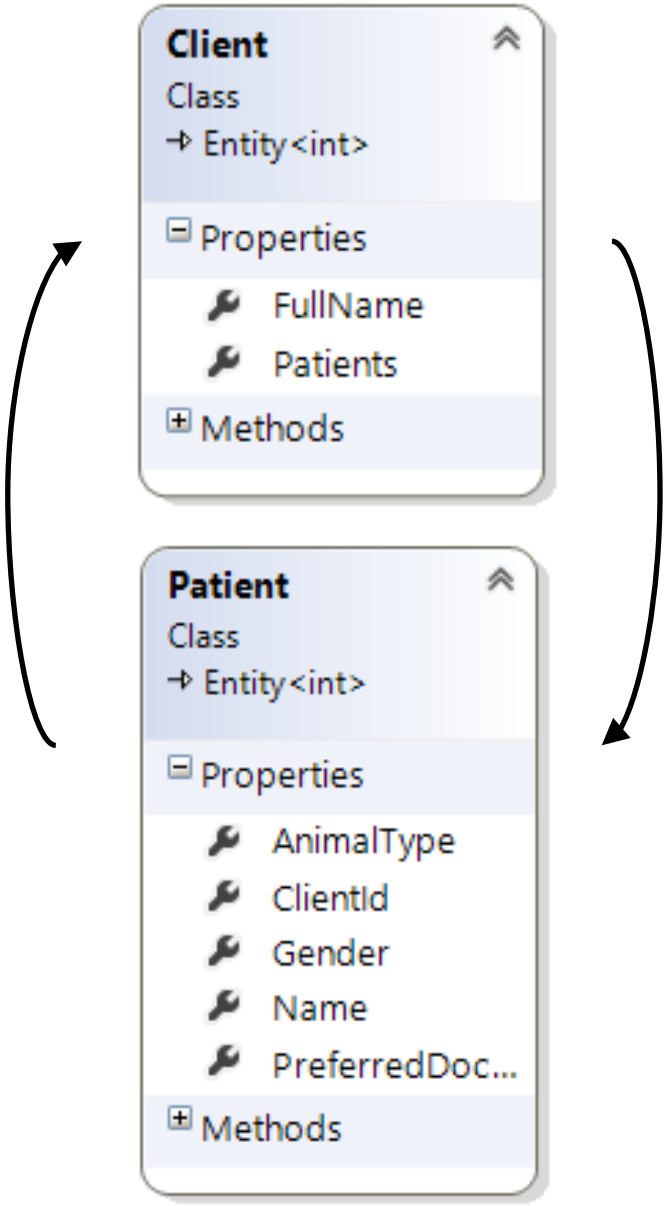
Evans,
Domain-Driven Design, p. 65

Entities in the Appointment Scheduling Context



Implementing Entities in Code

Relationships



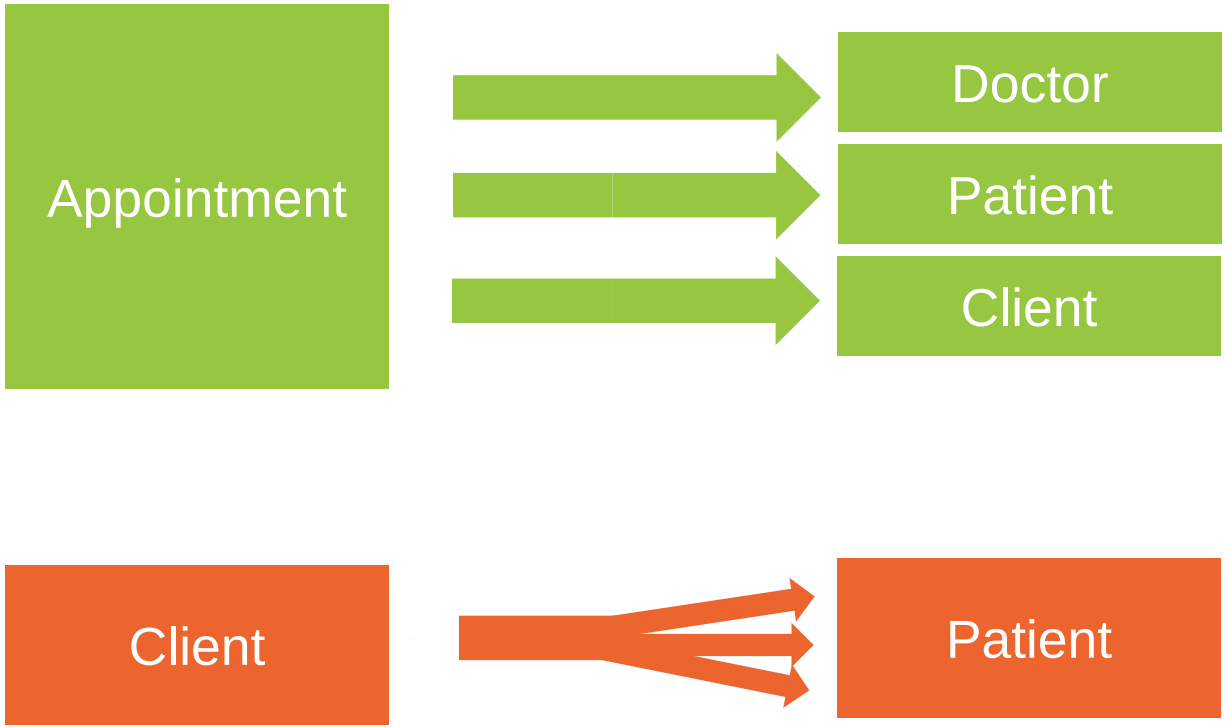
> A bidirectional association means that both objects can be understood only together. When application requirements do not call for traversal in both directions, adding a traversal direction reduces interdependence and simplifies the design.

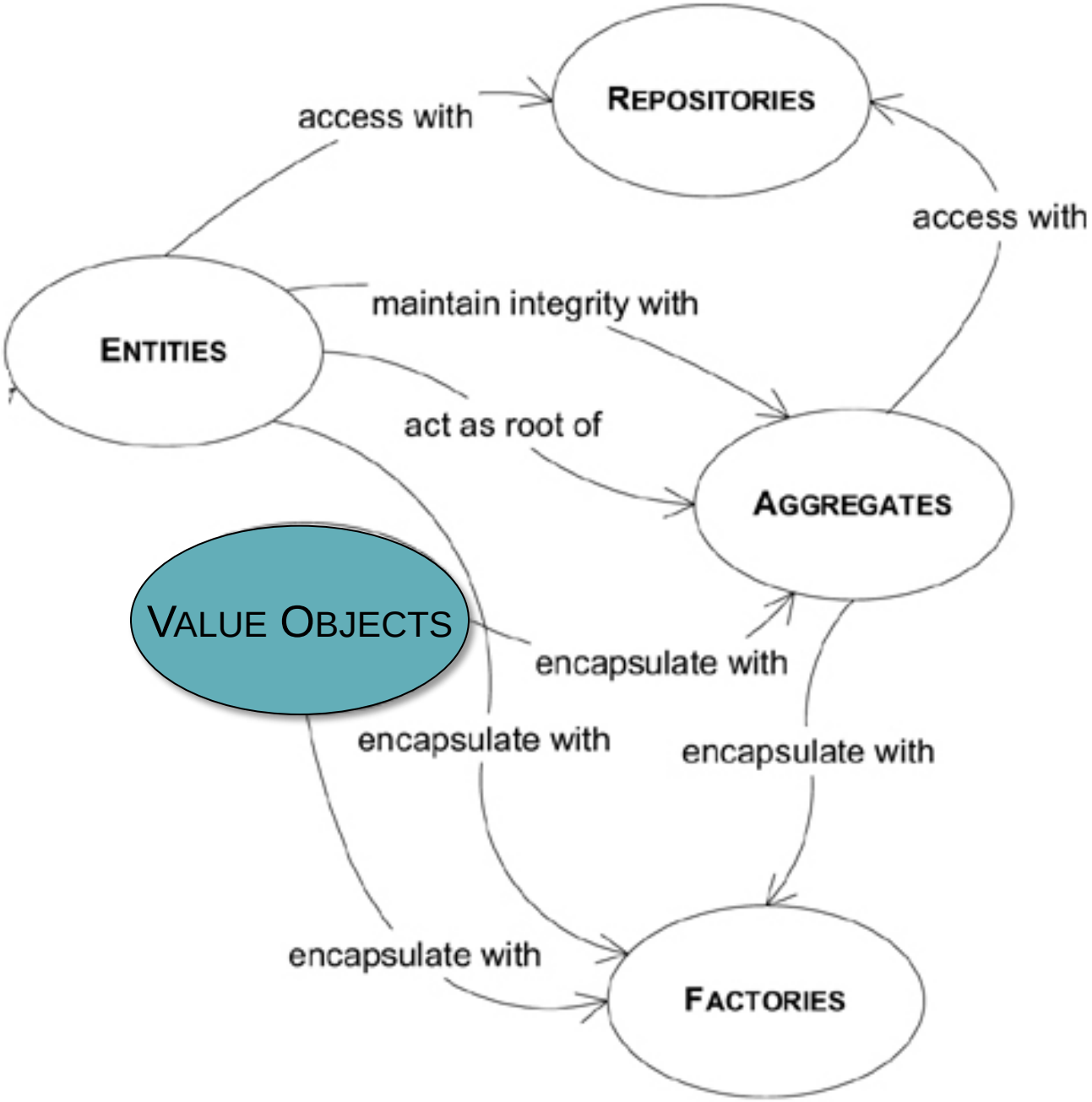
— Eric Evans

Domain-Driven Design

Start with One-Way Relationships

Uni-Directional Associations




REPOSITORIES
access with
access with
ENTITIES
maintain integrity with
act as root of
AGGREGATES
VALUE OBJECTS
encapsulate with
encapsulate with
encapsulate with
encapsulate with
FACTORIES

Value Object

Measures, quantifies, or describes a thing in the domain.

Identity is based on composition of values

Immutable

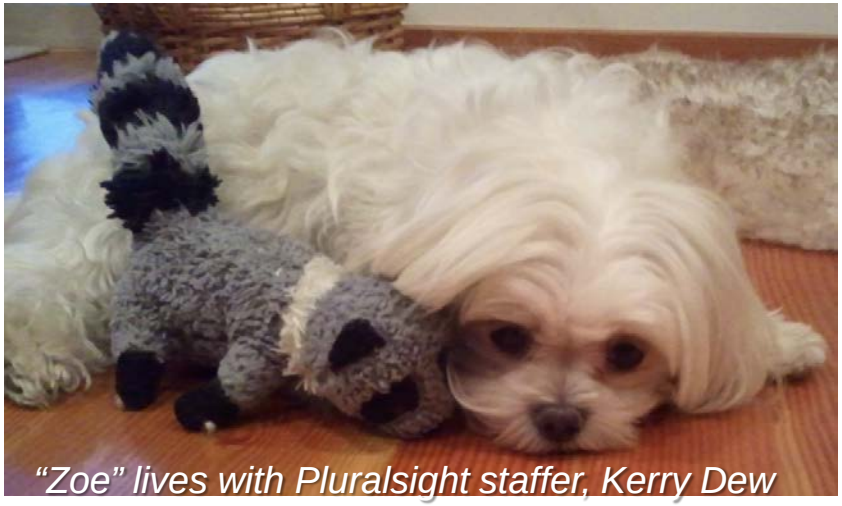
Compared using all values

No side effects

String: Our Favorite Value Object

C AR

DOG



"Zoe" lives with Pluralsight staffer, Kerry Dew

Company Worth: $50,000,000

$ 50,000,000

```
Company (Entity)
   ID (guid): 9F63CE8D-9F1E-45E0-85AB-C098CC15F8E6
   Worth Unit (string): “US Dollar”
   Worth Amount (decimal): 50000000
```

```
Company (Entity)
   ID (guid): 9F63CE8D-9F1E-45E0-85AB-C098CC15F8E6
   Worth { Worth (Value Object)
           Monetary Unit (string) “U.S. Dollar”
           Amount (decimal): 50000000
```

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

Patient Appointment

10:00 am Jan 4, 2014 – 11:00 am Jan 4, 2014

Staff Meeting

2:00 pm Feb 1, 2014 – 3:15 pm Feb 1, 2014

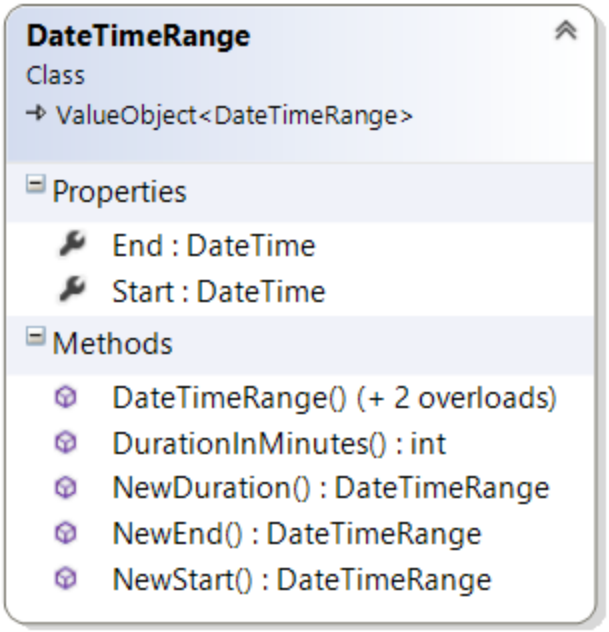
```
public class DateTimeRange
{
    public DateTimeRange(DateTime start, DateTime end)
   {
       Start=start;
       End=end;
    }
    public DateTime Start { get; private set; }
    public DateTime End { get; private set; }
    …
}
```

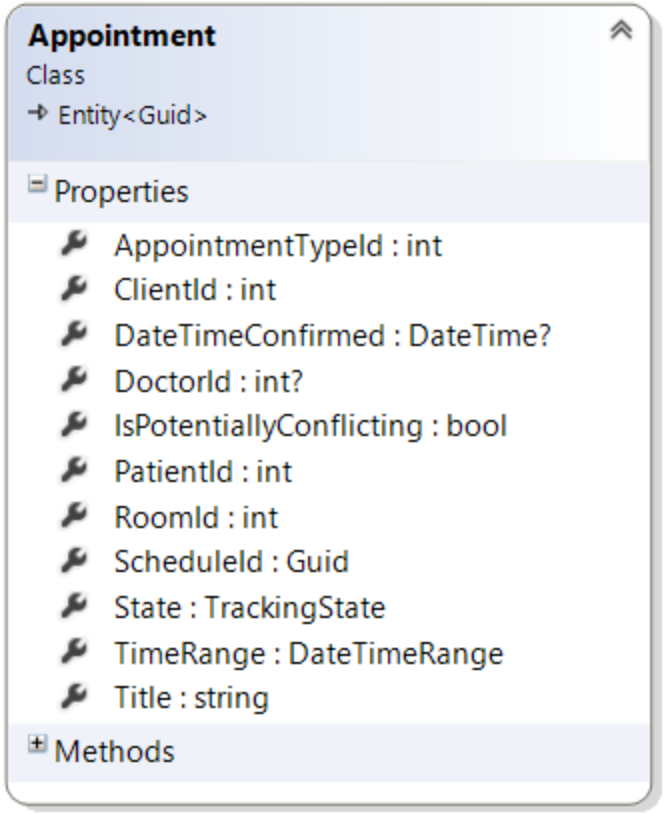
> It may surprise you to learn that we should strive to model using Value Objects instead of Entities wherever possible. Even when a domain concept must be modeled as an Entity, the Entity's design should be biased toward serving as a value container rather than a child Entity container.

— Vaughn Vernon

Implementing Domain-Driven Design

Our DateTimeRange Value Object





Implementing Value Objects in Code

Domain Services

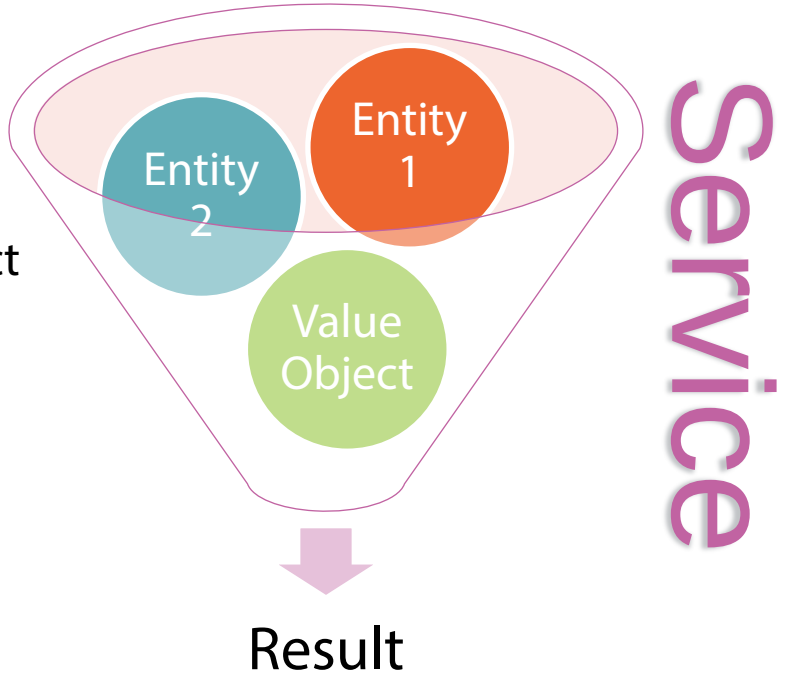
Important operations that don't belong to a particular Entity or Value Object

---

Good Domain Services:

- Not a natural part of an Entity or Value Object
- Have an interface defined in terms of other domain model elements
- Are stateless (but may have side effects)

Live in the Core of the application



Examples of Services in Different Layers

UI Layer
& Application Layer

- Message Sending
- Message Processing
- XML Parsing
- UI Services

Domain
("Application Core")

- Transfer Between Accounts
- Process Order

Infrastructure

- Send Email
- Log to a File

Glossary of Terms from this Module

Anemic Domain Model

Model with classes focused on state management.
Good for CRUD.

Rich Domain Model

Model with logic focused on behavior, not just state.
Preferred for DDD.

Entity

A mutable class with an identity (not tied to it's property values) used for tracking and persistence.

Immutable

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

Glossary of Terms from this Module

Value Object

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

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)

D

is for

DOMAIN

References

Books

Domain-Driven Design http://amzn.to/1kstiRg

Implementing Domain-Driven Design http://amzn.to/1dgYRY3

Web

Jimmy Bogard – Services in DDD - http://bit.ly/1ifravE

DomainLanguage.com

On Pluralsight:

- SOLID Principles of OO Design - bit.ly/solid-smith

Thanks!

Julie Lerman

TheDataFarm.com

Twitter: @julielerman

Steve Smith

Ardalis.com

Twitter: @ardalis

To Teach Is To Learn Twice