Tactical Design

Tactical Design Patterns

Tactical design Intro

Entity vs Value Object

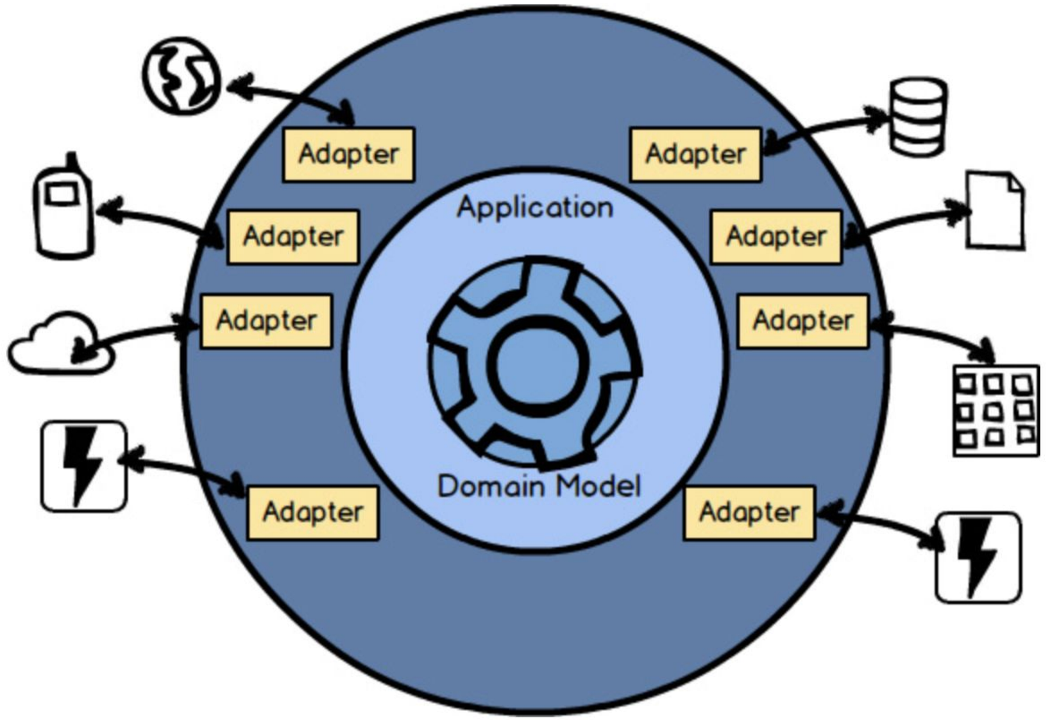
Aggregates

Anemic and Rich Domain Models

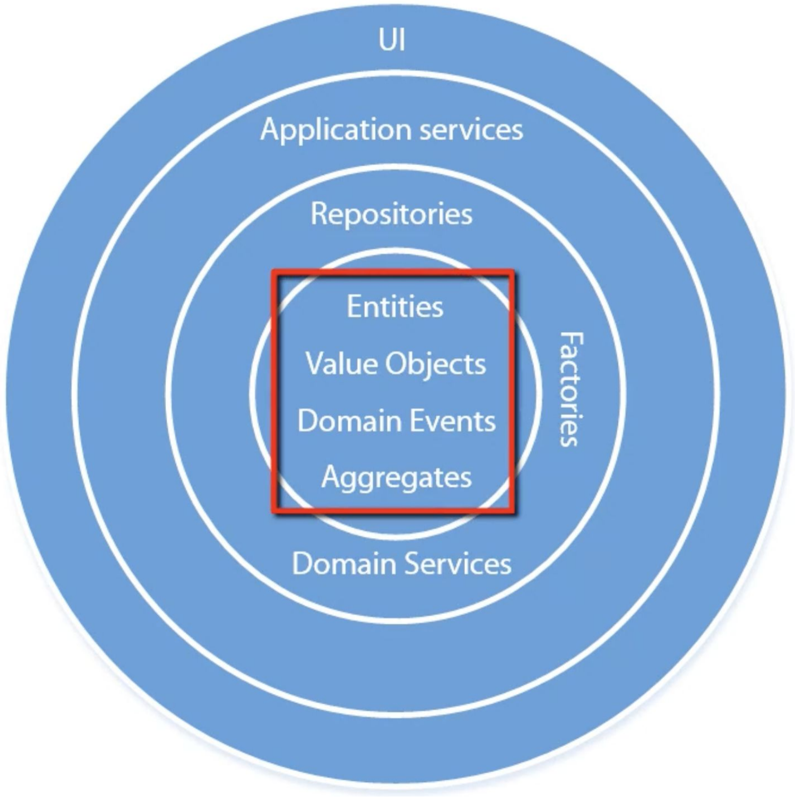
Repository Pattern

Domain Services, Application Services and Infrastructure Services

Intro


Adapter
Adapter
Adapter
Adapter
Adapter
Adapter
Adapter
Adapter
Application
Domain Model

Intro


UI
Application services
Repositories
Entities
Value Objects
Domain Events
Aggregates
Factories
Domain Services

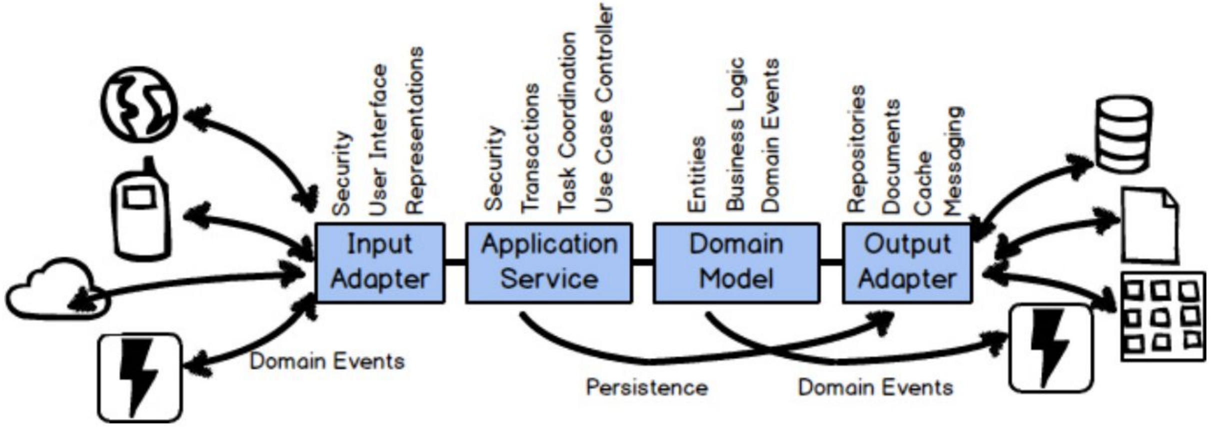
Intro

Input Adapters, such as user interface controllers, REST endpoints, and message listeners (Kafka, RabbitMQ)

Application Services that orchestrate use cases and manage transactions

The domain model that we've been focusing on

Output Adapters such as persistence management and message senders



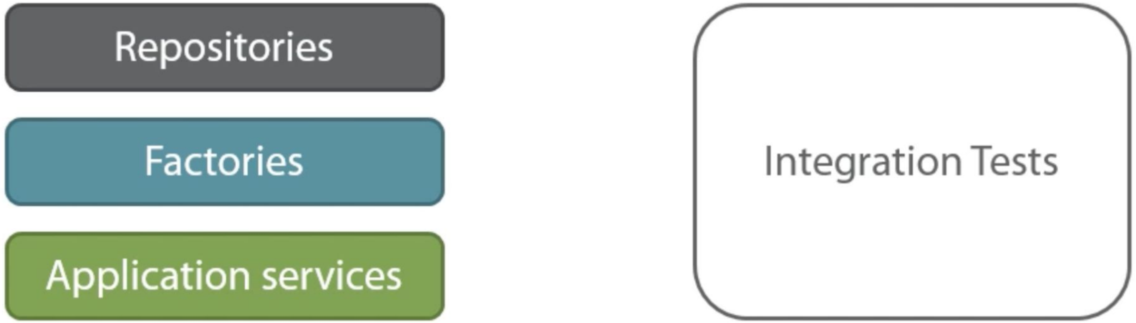
Intro

Isolation

Entity | Domain Event | Value Object | Aggregate

Domain knowledge	✓
Persistence	✗
Construction	✗
Mapping to the database	✗

Intro


Repositories
Factories
Application services
Integration Tests

Entities vs Value Objects

Entities

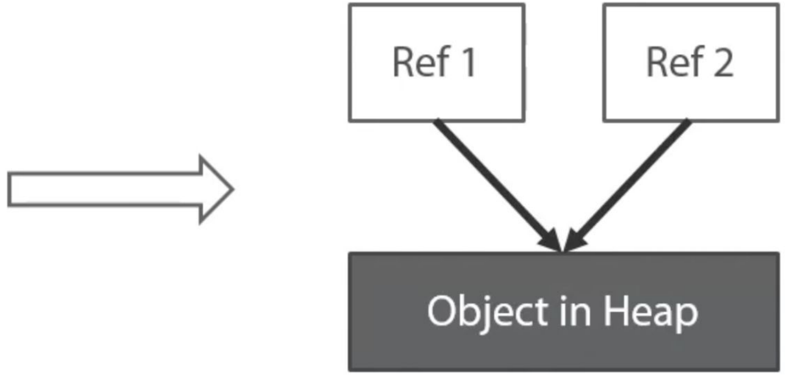
- ☐ Have inherent Identity
- ☐ Mutable

Value Objects

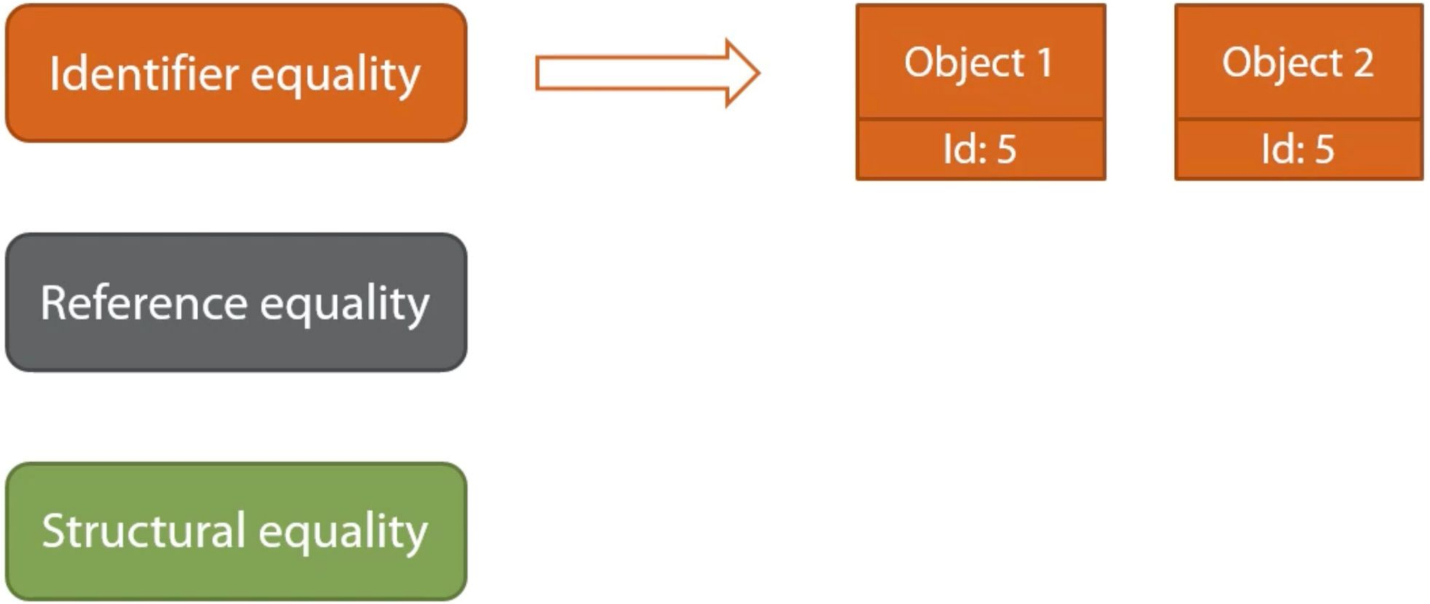
- ☐ Don't have an Id field
- ☐ Can be treated interchangeably
- ☐ Immutable

Equality Types

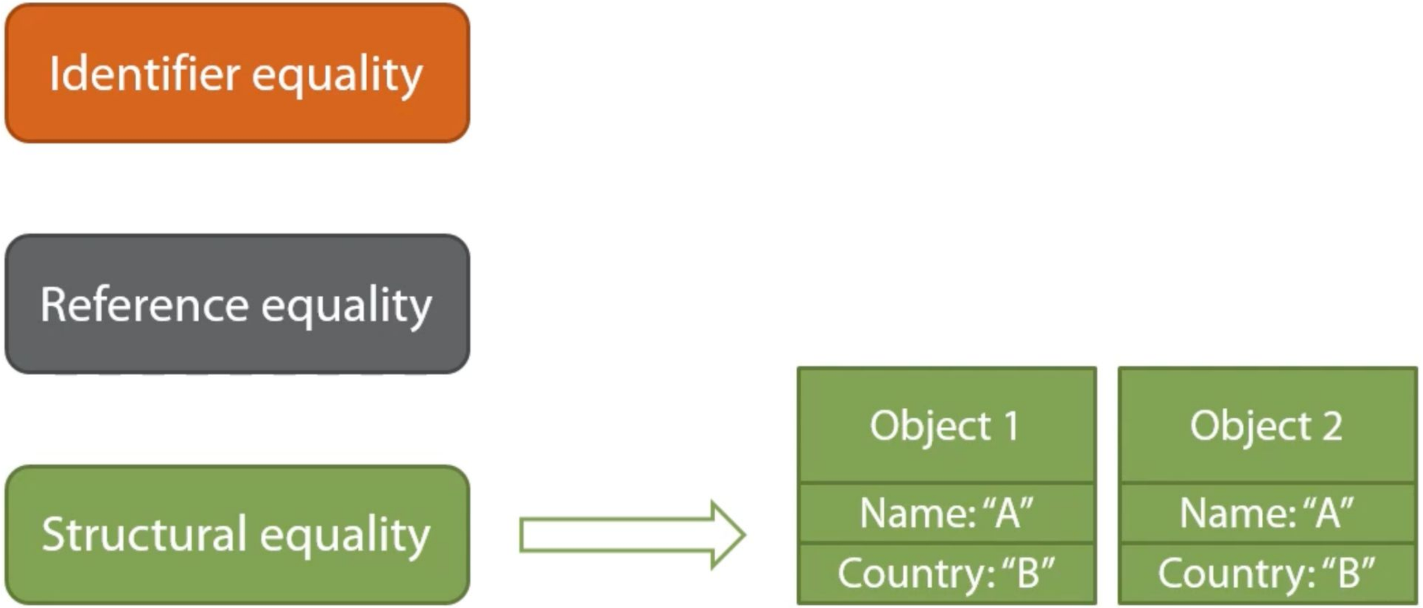
- Identifier equality
- Reference equality
- Structural equality



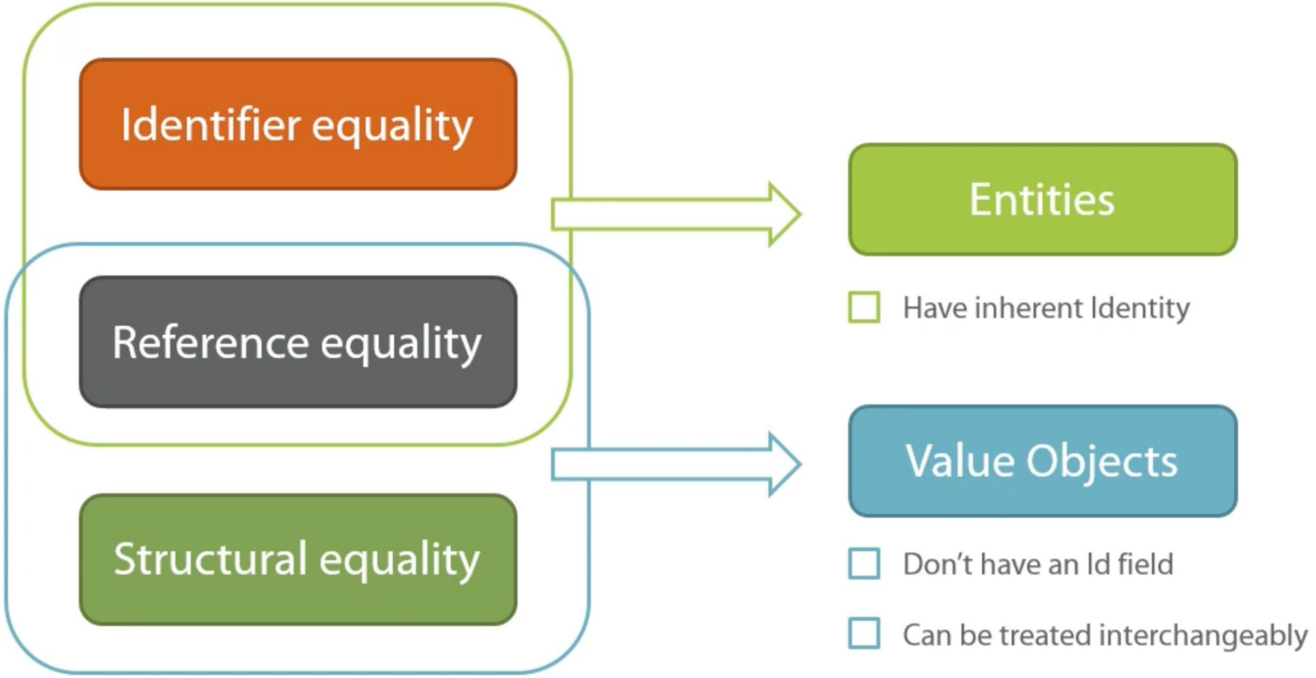
Equality Types



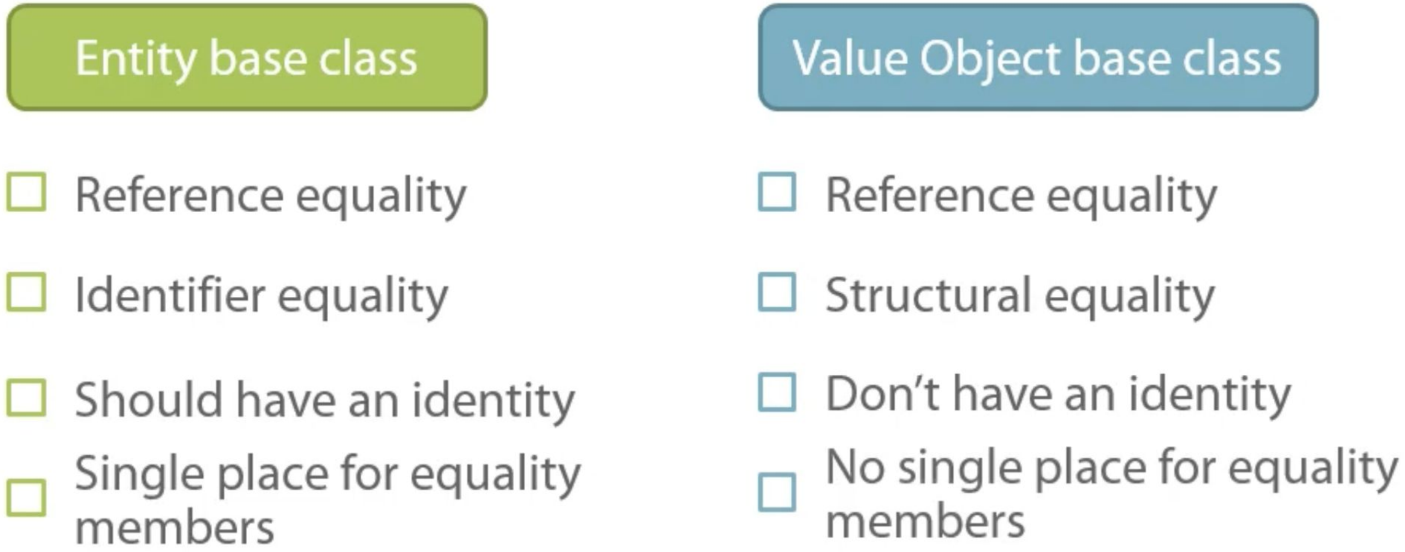
Equality Types



Equality for Entities and Value Objects



Base Class Design



Entity base class	Value Object base class
☐ Reference equality	☐ Reference equality
☐ Identifier equality	☐ Structural equality
☐ Should have an identity	☐ Don’t have an identity
☐ Single place for equality members	☐ No single place for equality members

Entity

Each Entity has a **unique identity** in that you can distinguish its individuality from among all other Entities.

- Entity will be **mutable**
- The main thing that separates an Entity from other modeling tools is its uniqueness

Entity

Client : BaseEntity<int>

- FullName
- Patients

Patient : BaseEntity<int>

- AnimalType
- ClientId
- Gender
- Name
- PreferredDoctor

Appointment : BaseEntity<Guid>

- ClientId
- DoctorId
- PatientId
- RoomId
- StartEndTime

Doctor : BaseEntity<int>

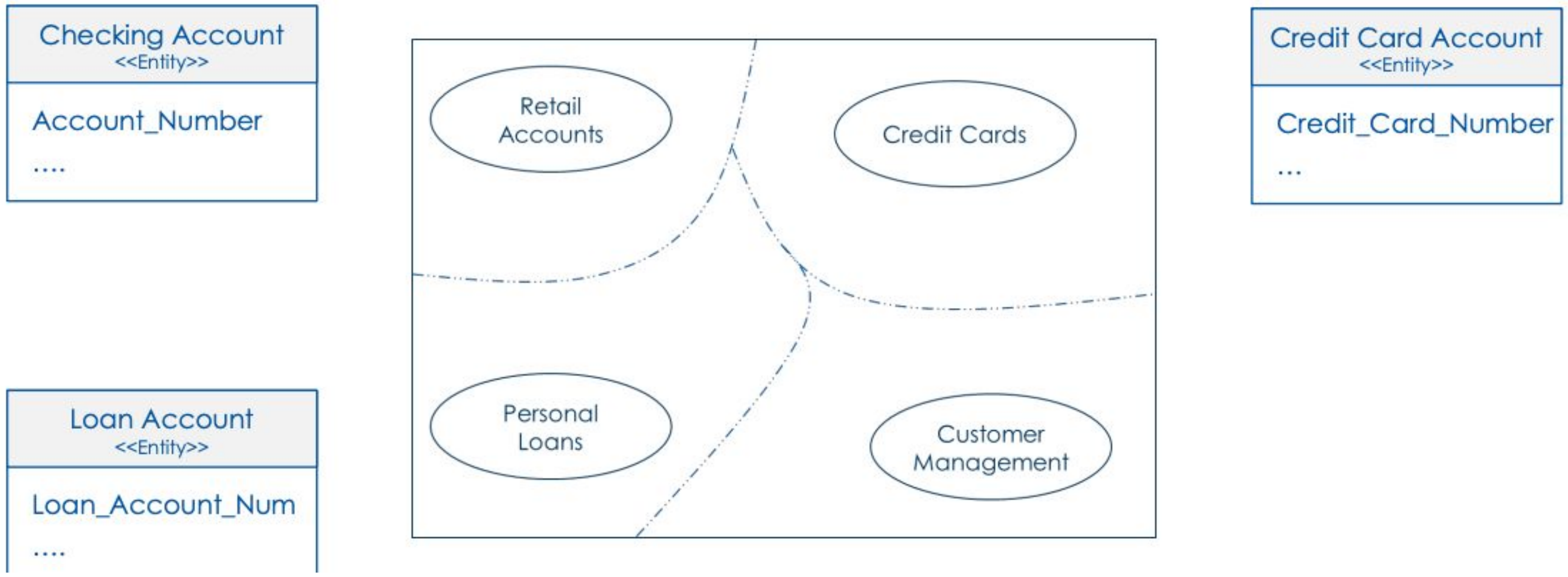
- Name

Room : BaseEntity<int>

- Name

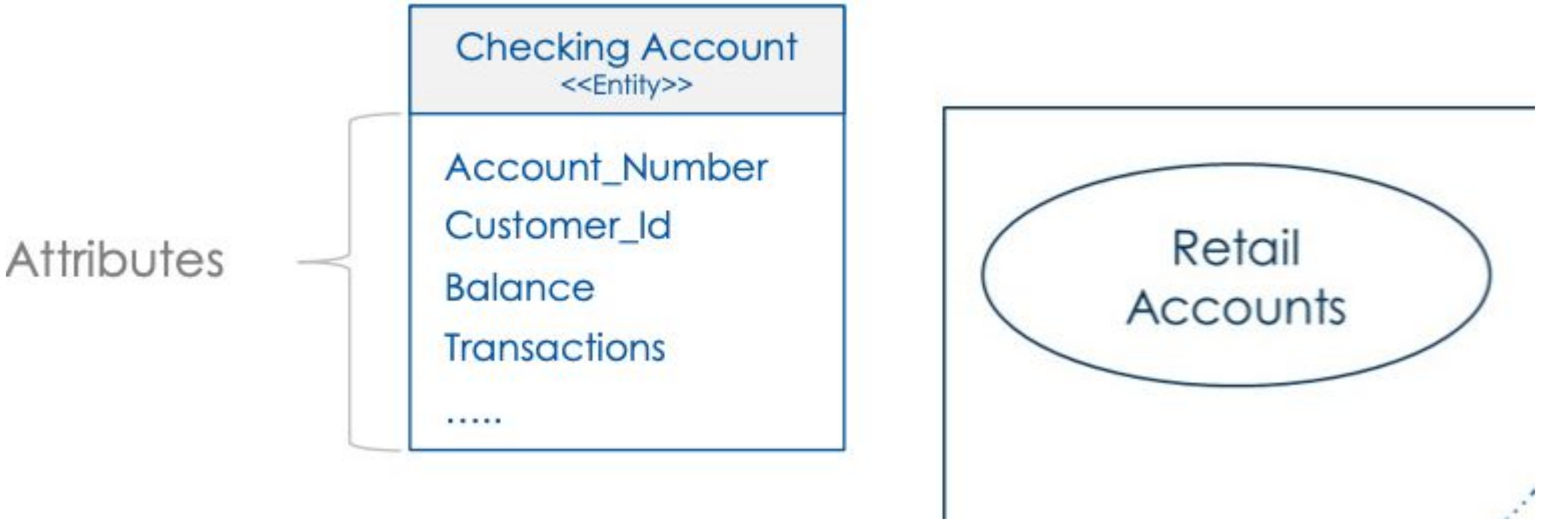
Entity

An entity is uniquely identified within a Bounded Context



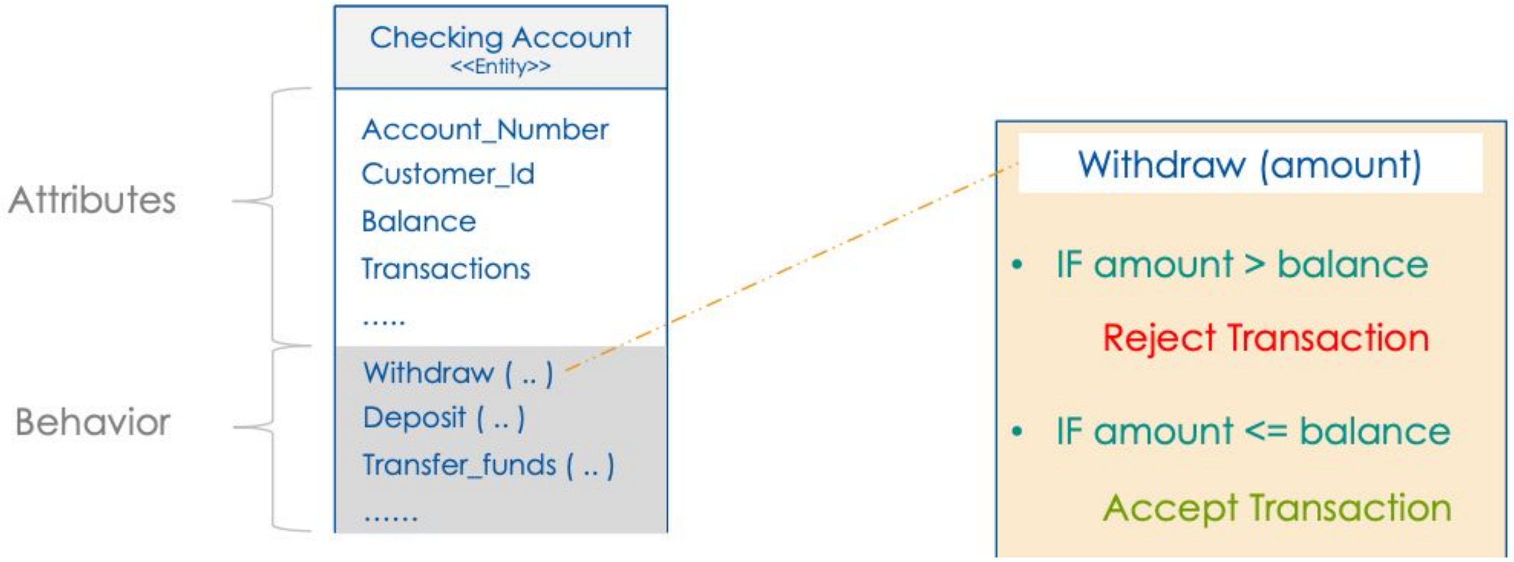
Entity

Entity attributes are defined as per the Ubiquitous Language



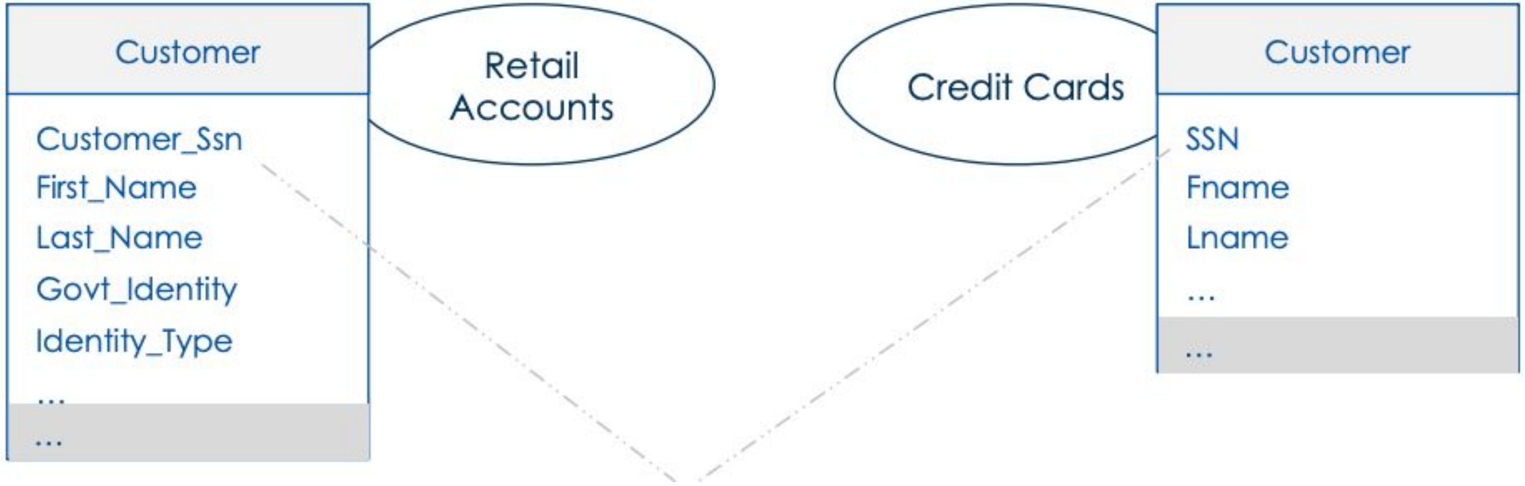
Entity

Entity's state is managed by way of operations (business logic)



Entity

An entity is meaningful within a Bounded Context

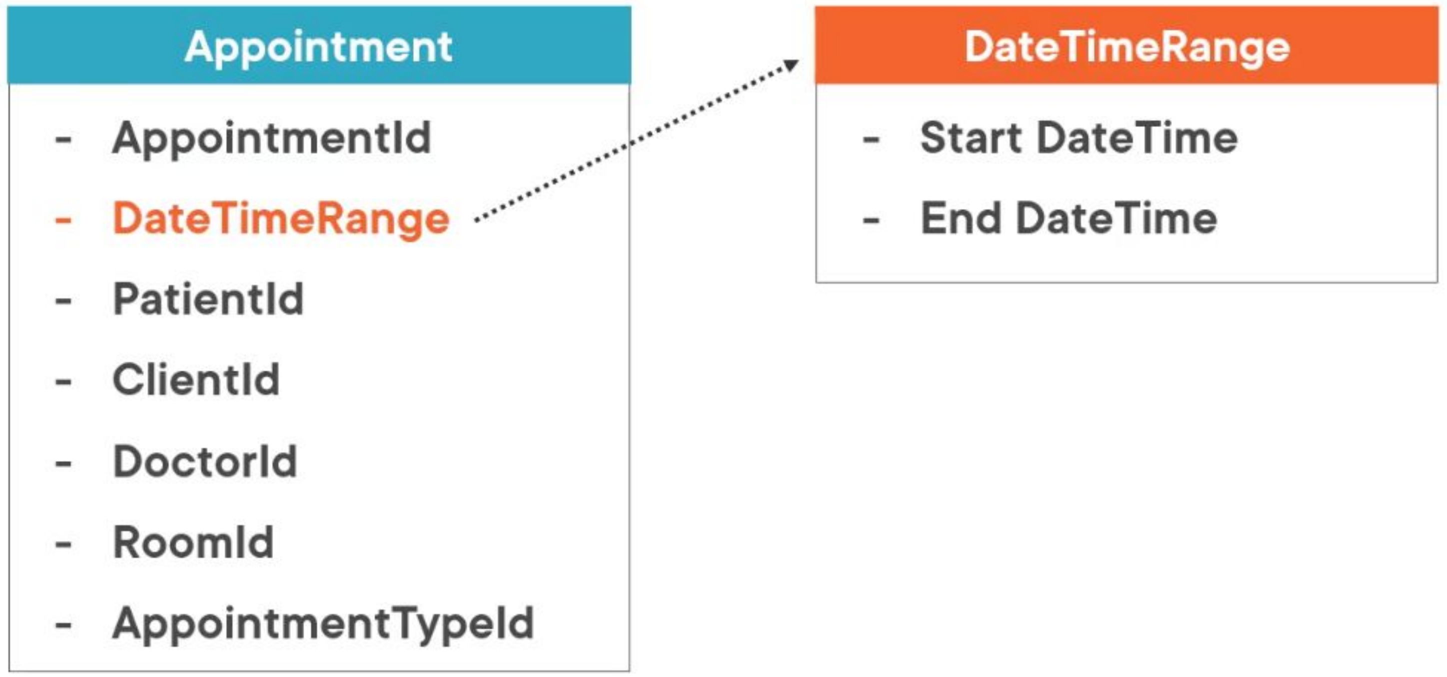


Value Object

A Value Object, or simply a Value, models an immutable conceptual whole.

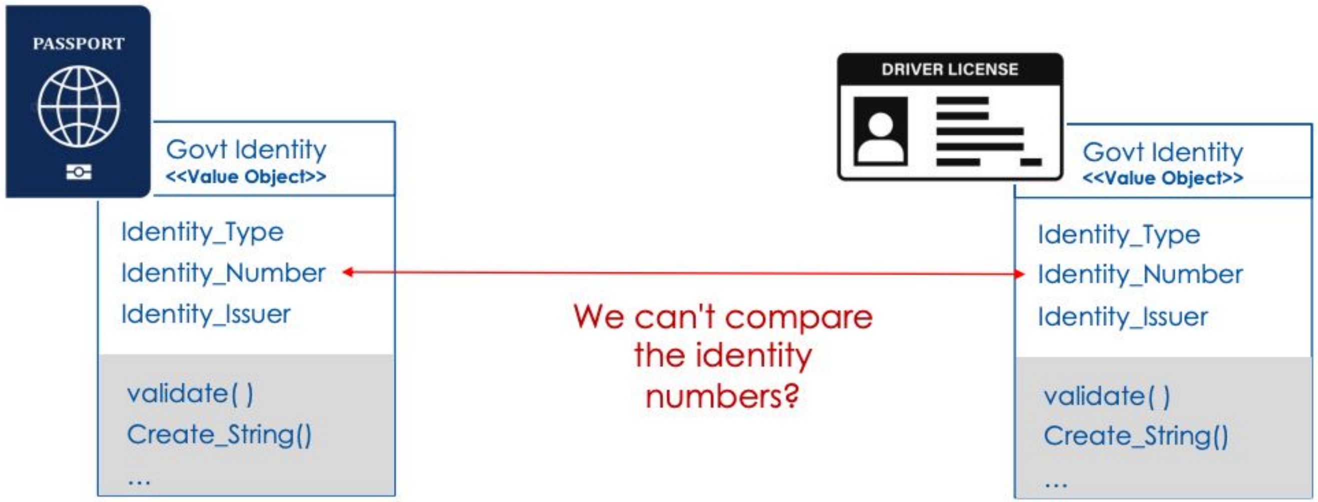
- It does not have a unique identity
- It is often used to describe, quantify, or measure an Entity

Value Object



Value Object

Equality check is based on attributes

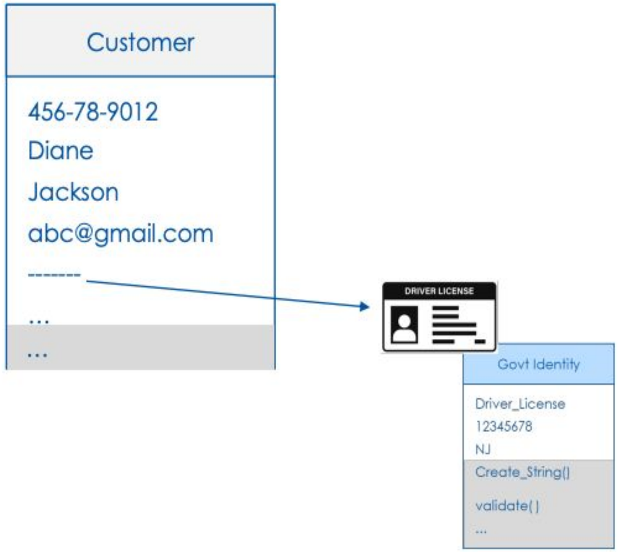


Value Object

Value objects are immutable

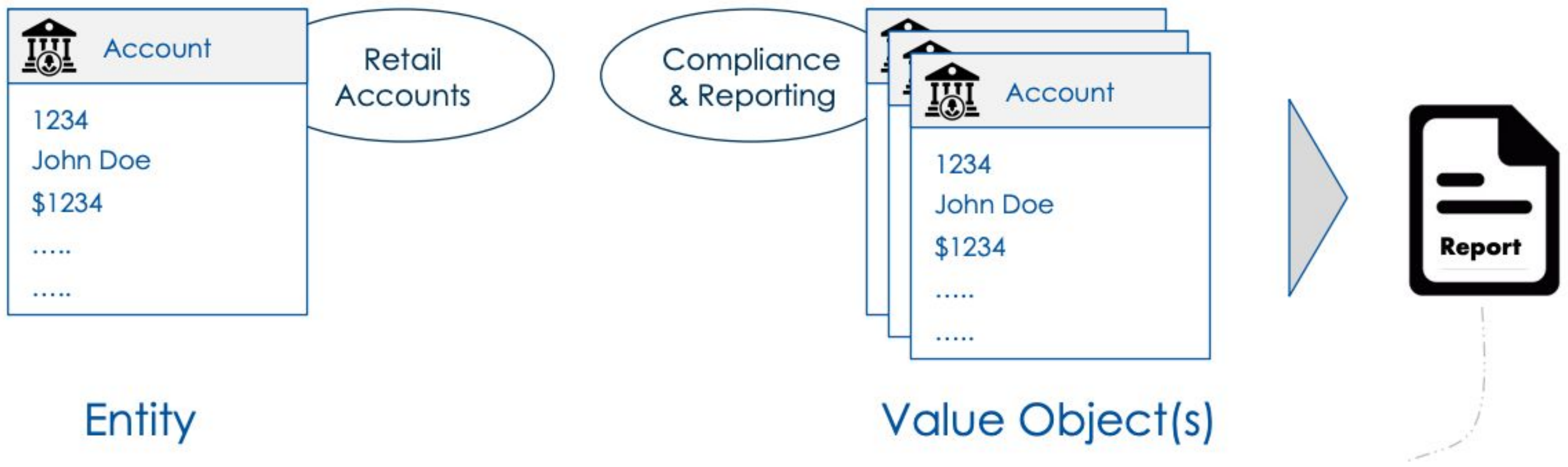
Value objects have meaning only in the context of an Entity

Persisted as part of Entity Object



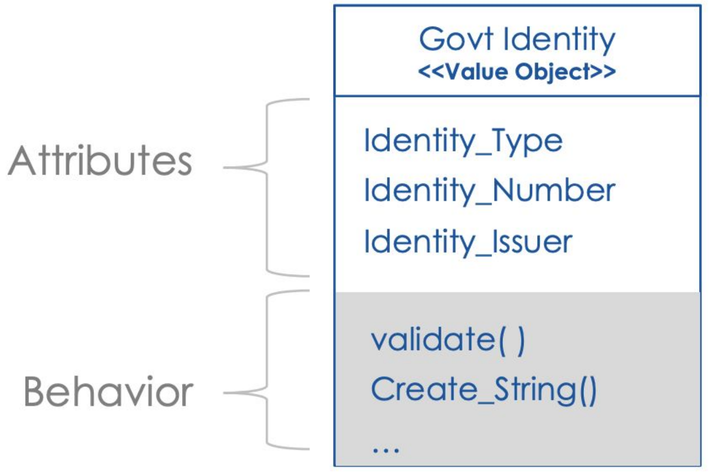
Value Object

Value objects in one BC may be an Entity in another BC



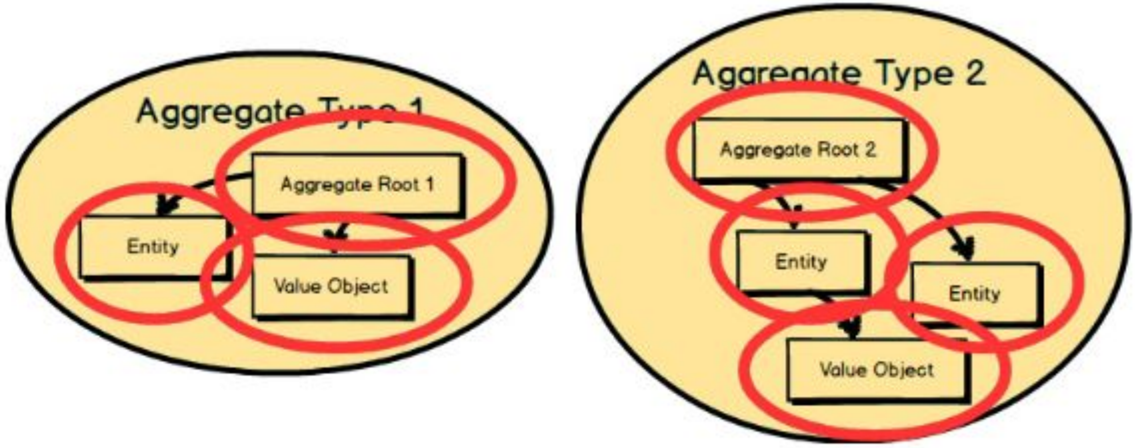
Value Object

Like the Entities, the Value Objects have attributes & behavior



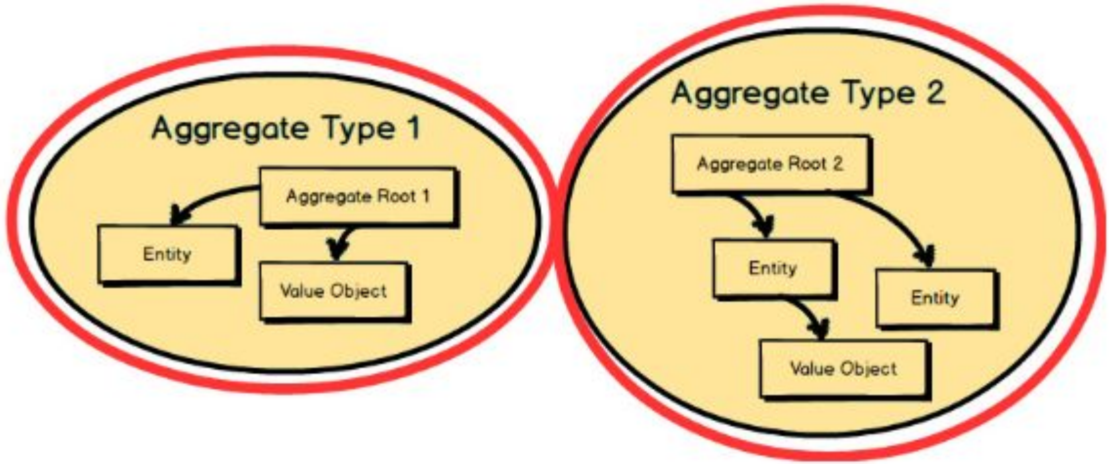
Aggregates

Each Aggregate is composed with Entities and Value Objects. The name of the Root Entity is the Aggregate's conceptual name. You should choose a name that properly describes the conceptual whole that the Aggregate models.



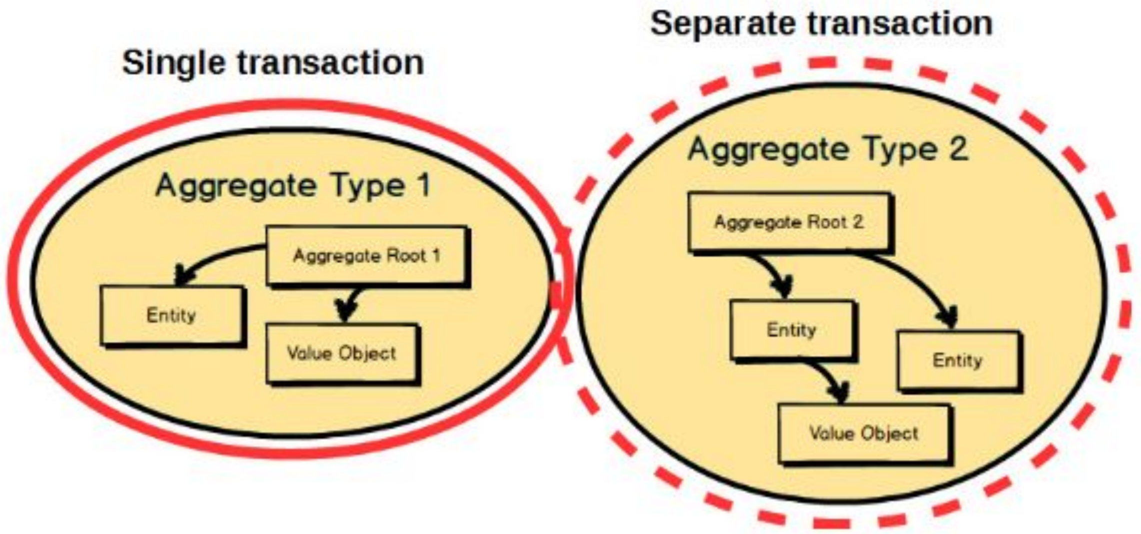
Aggregates

Each Aggregate forms a **transactional consistency boundary**. If the Aggregate was not stored in a whole and valid state, the business operation that was performed would be considered incorrect according to business rules.



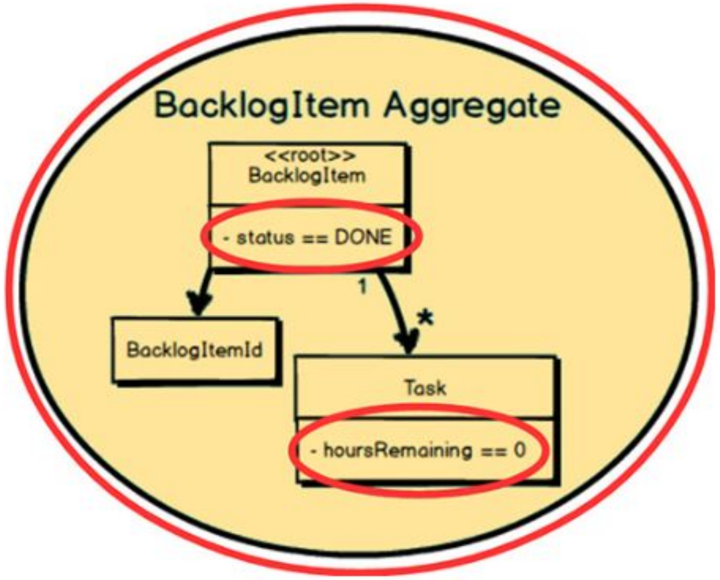
Aggregates

Aggregate design: modify and commit only one Aggregate instance in one transaction.



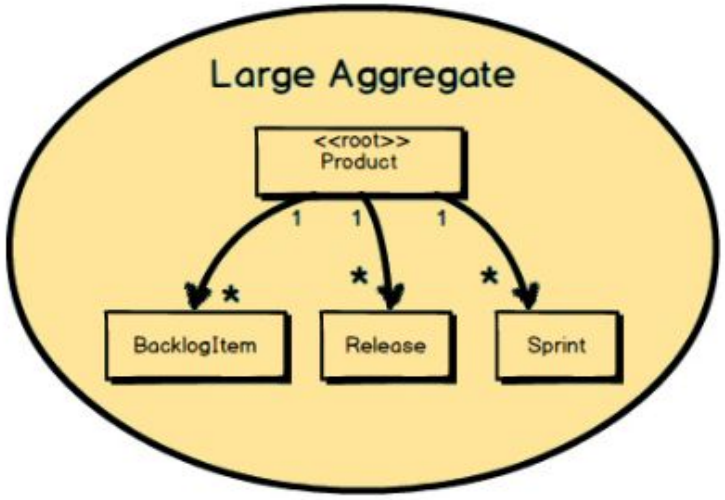
Aggregate Design Rules

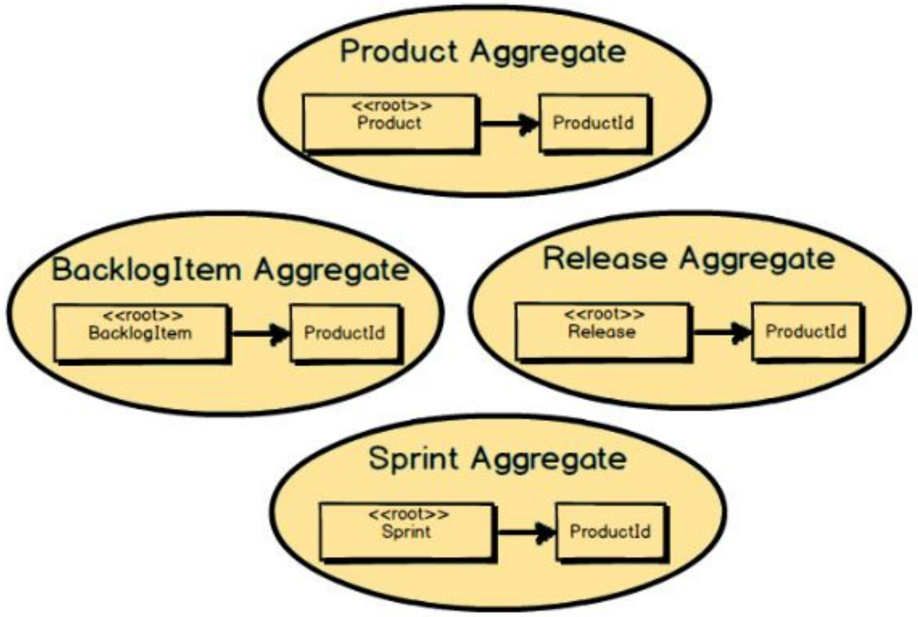
1- Protect Business Invariants inside Aggregate Boundaries



Aggregate Design Rules

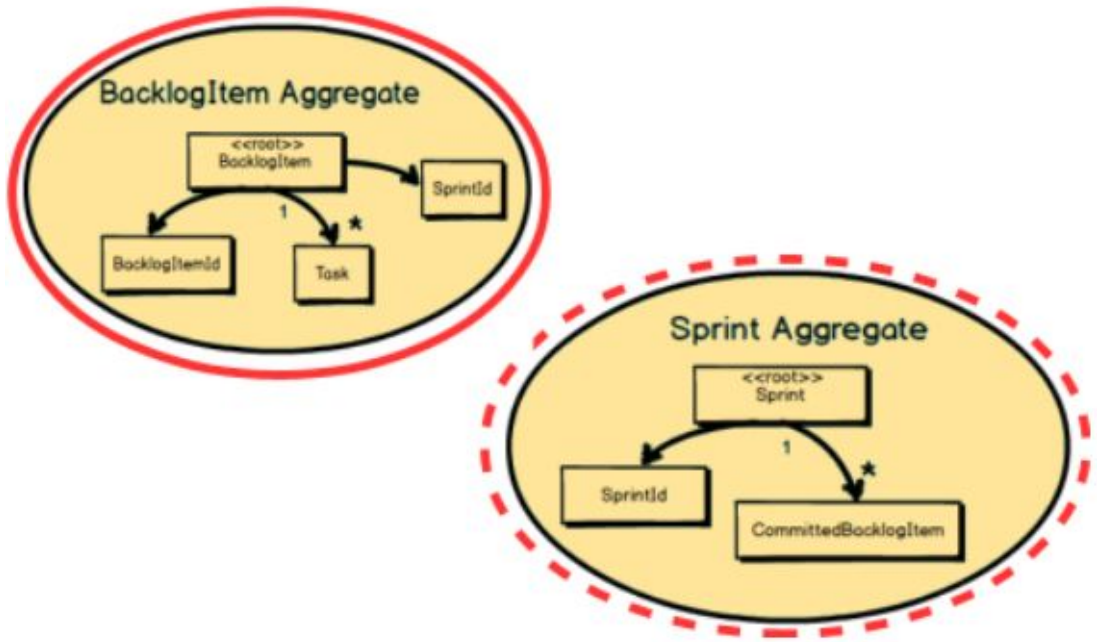
2- Design Small Aggregates; Single Responsibility Principle





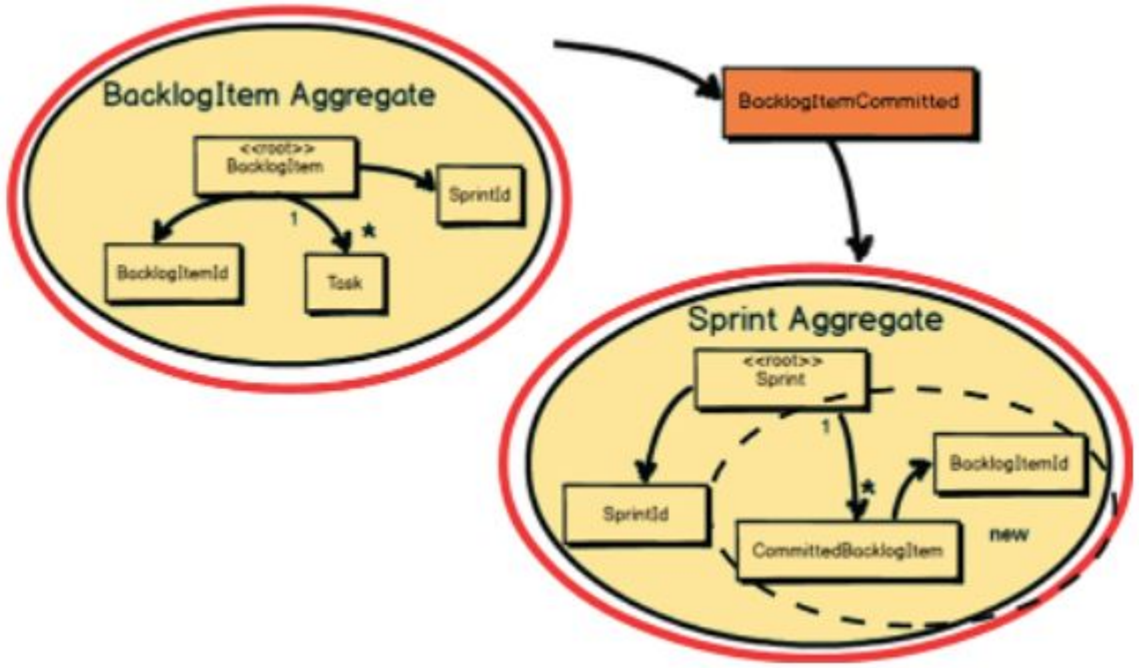
Aggregate Design Rules

3- Reference Other Aggregates by Identity Only



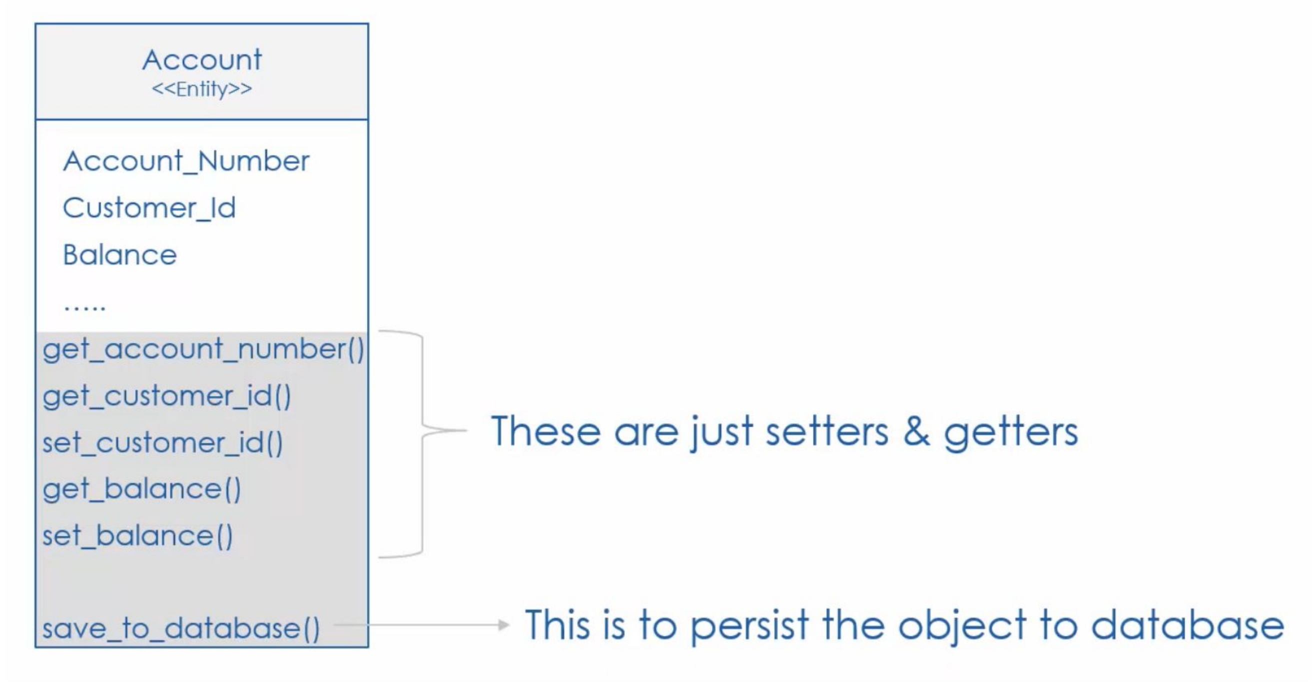
Aggregate Design Rules

4- Update Other Aggregates Using Eventual Consistency



Anemic and Rich Domain Models

Anemic Models

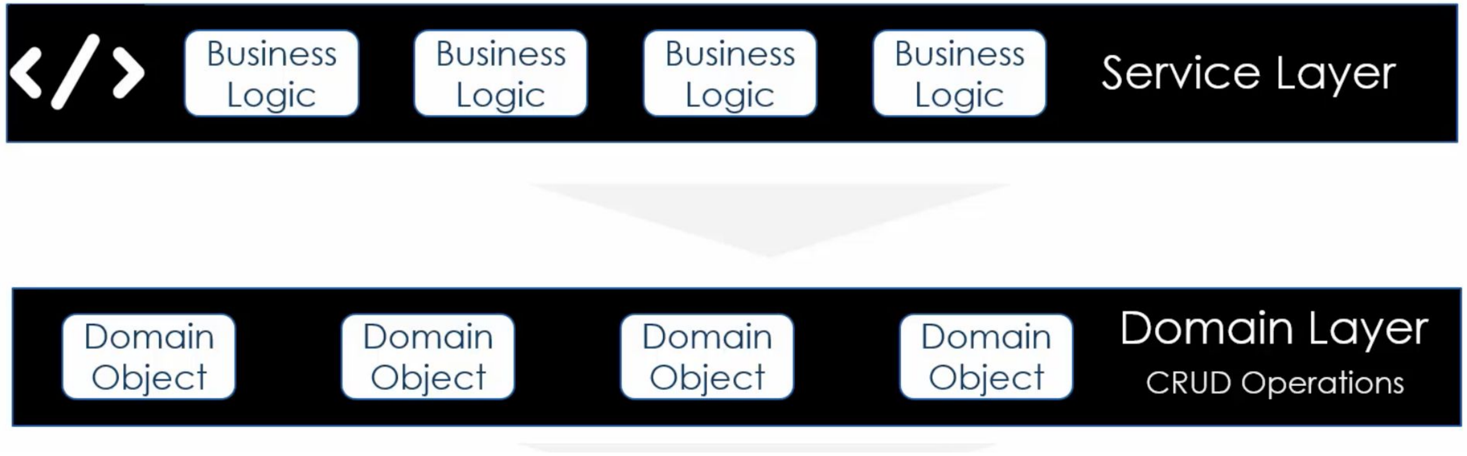


Anemic and Rich Domain Models

#1 Entities lack the behavior

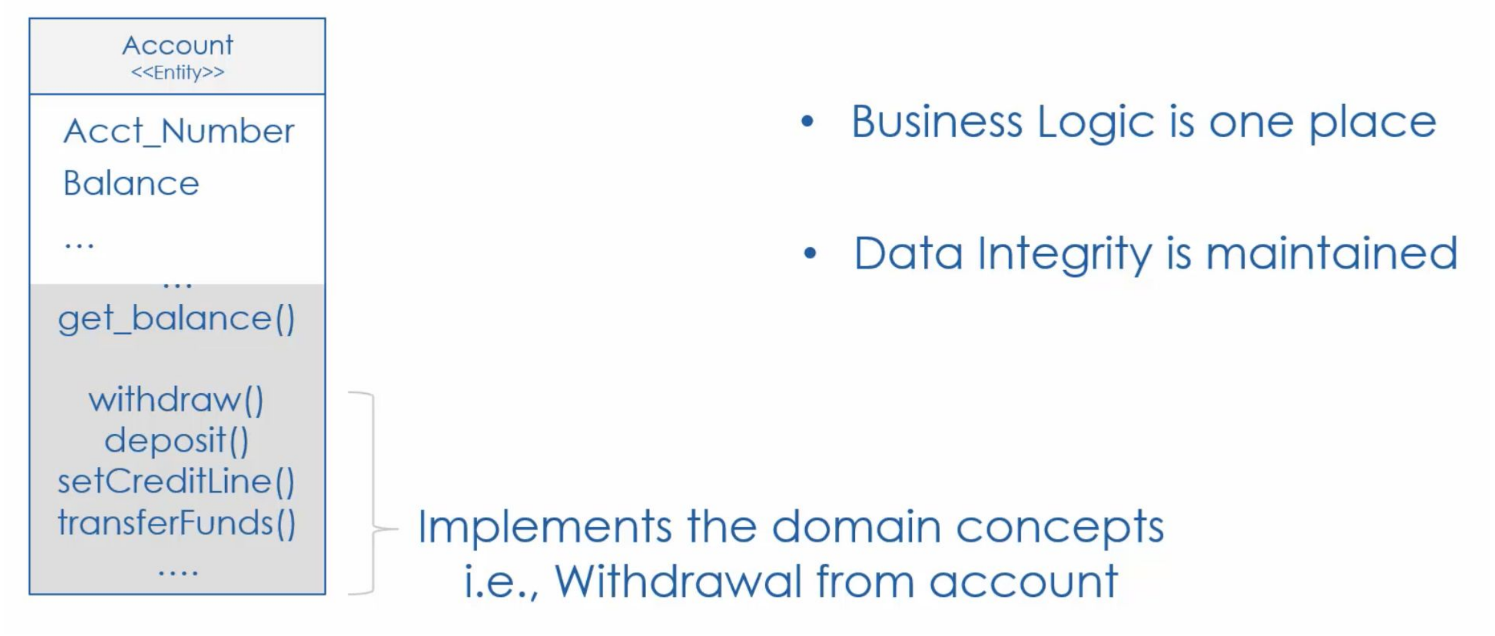
#2 Entity exposes functions ONLY for CRUD operations

#3 Business Logic is implemented outside the Domain Objects



Anemic and Rich Domain Models

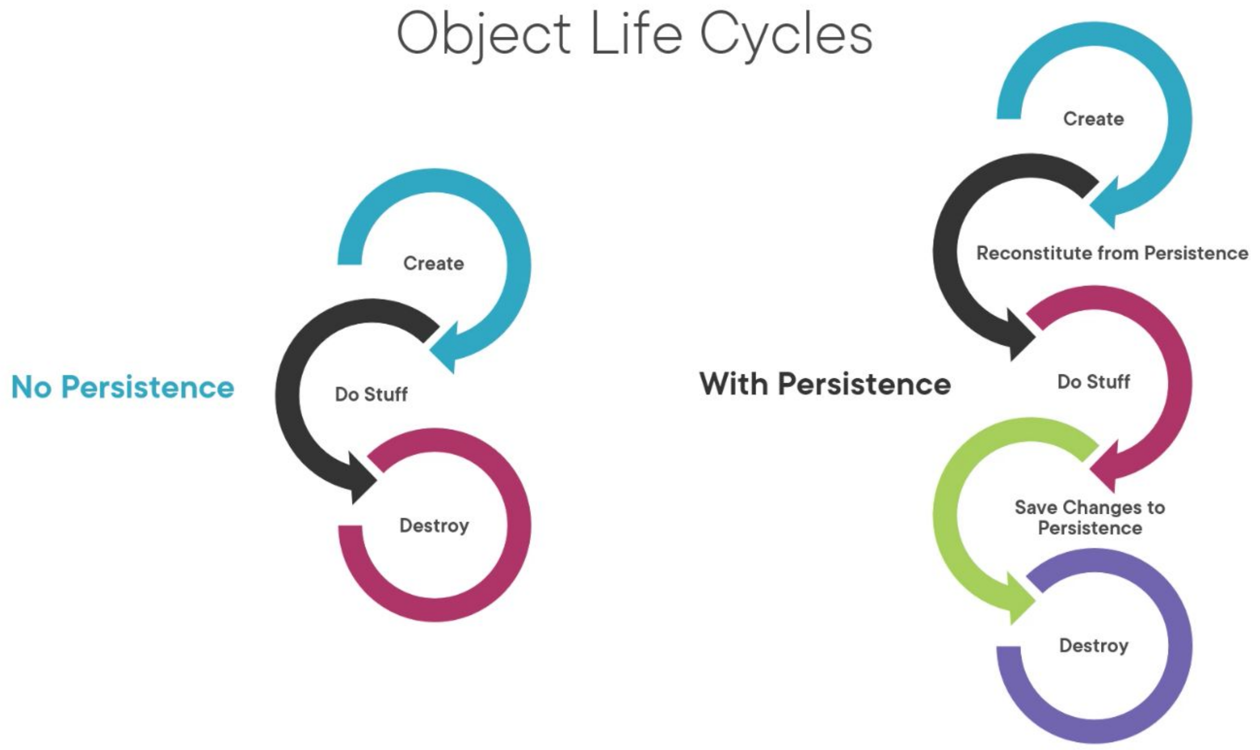
Rich Models



- Business Logic is one place
- Data Integrity is maintained

Implements the domain concepts
i.e., Withdrawal from account

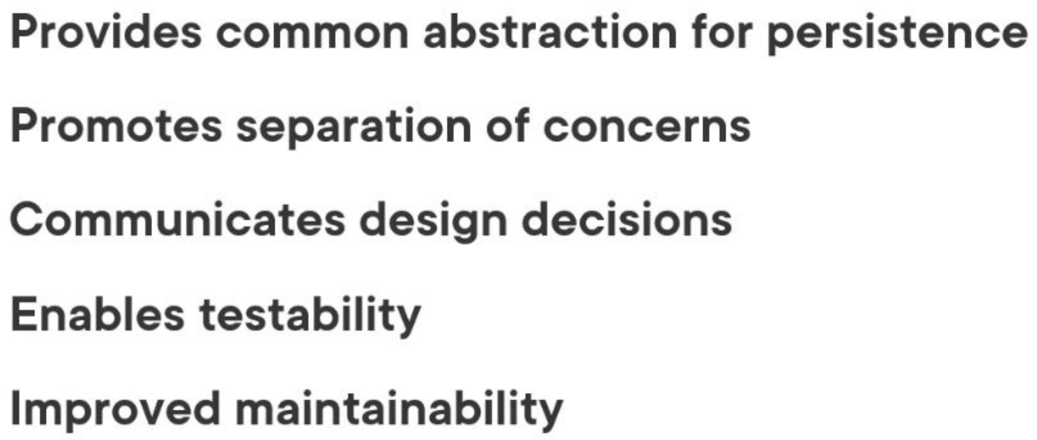
Repository Pattern



Repository Pattern

“It hides the storage level details needed for managing & querying the state of the Aggregate in the underlying data tier.”

Benefits:



Provides common abstraction for persistence

Promotes separation of concerns

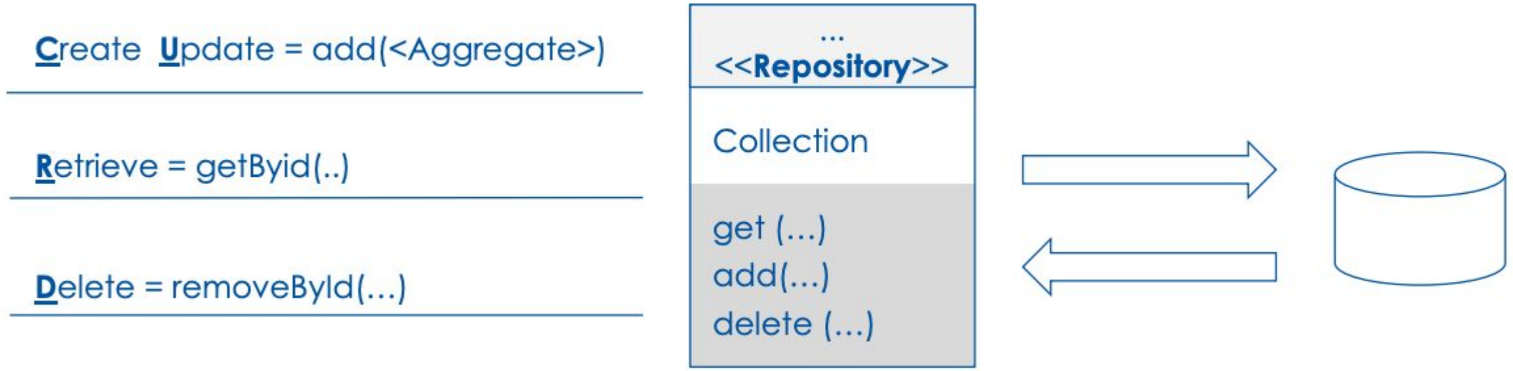
Communicates design decisions

Enables testability

Improved maintainability

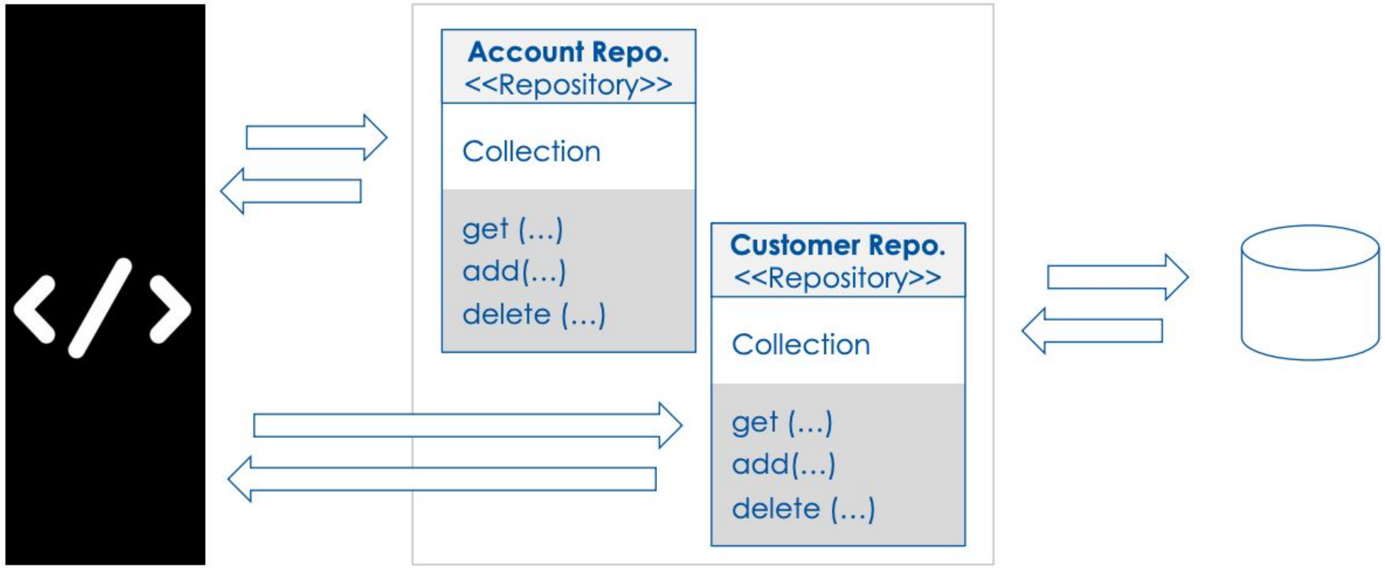
Repository Pattern

Manage | Query the state of aggregate



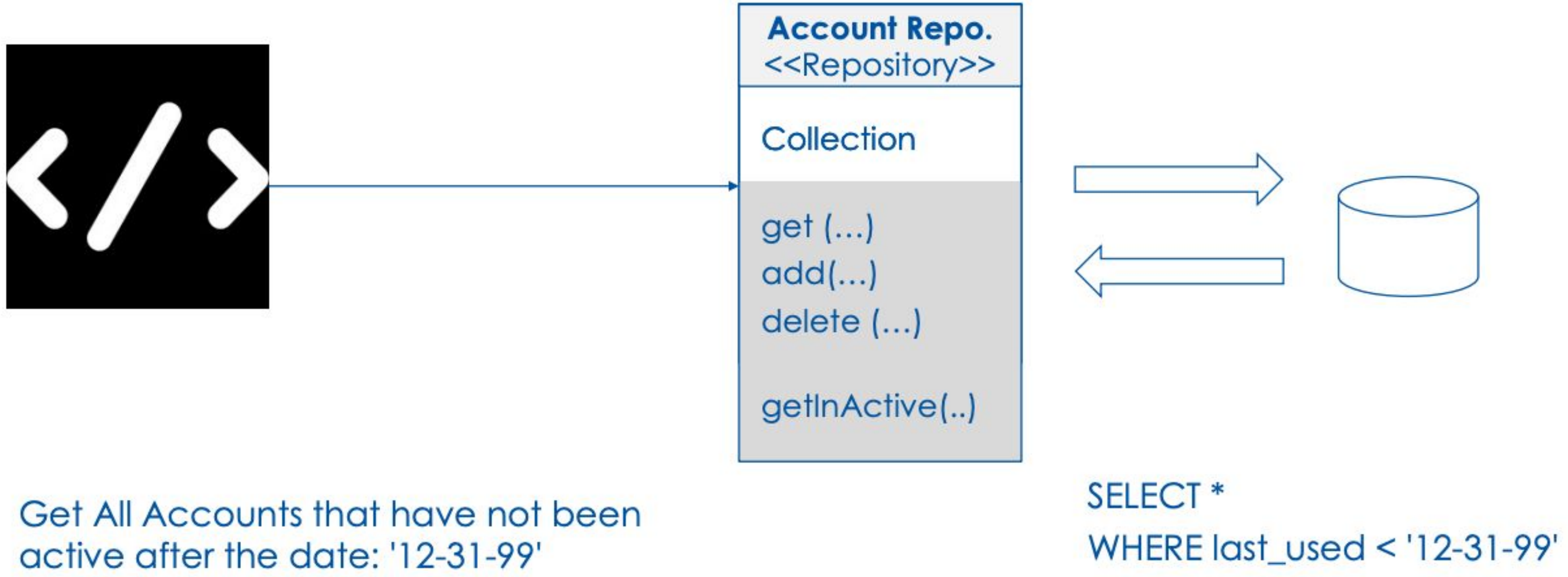
Repository Pattern

Created on per Aggregate basis



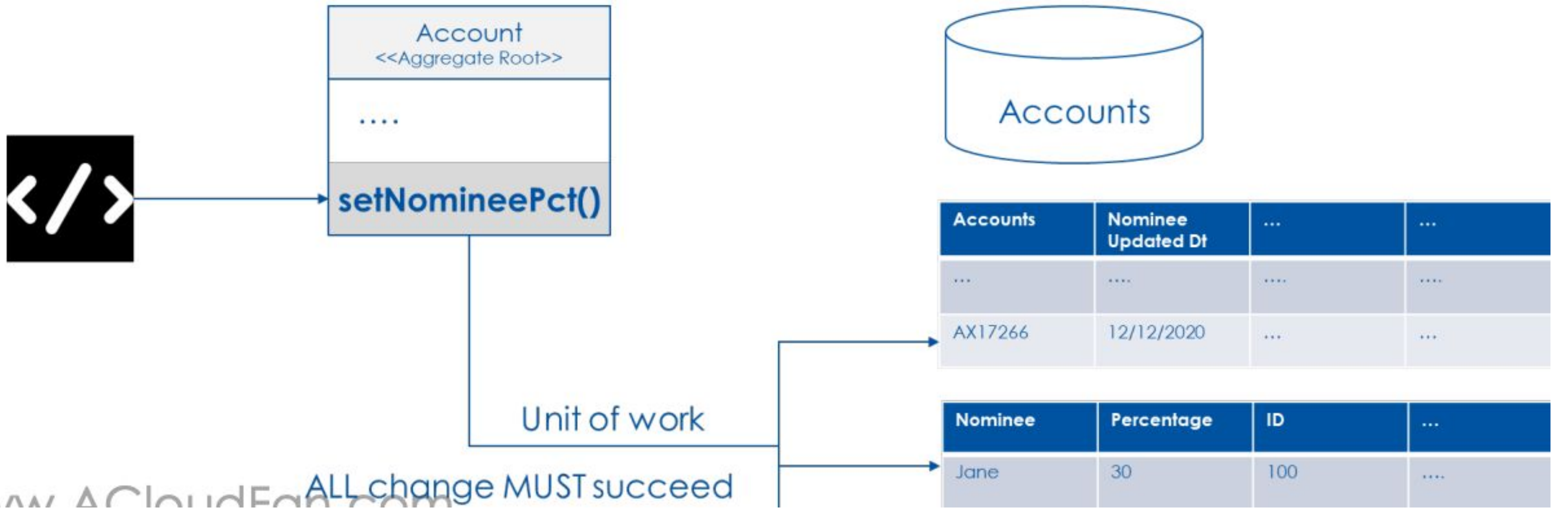
Repository Pattern

May expose higher level behavior | functions


Account Repo.
<<Repository>>
Collection
get (...)
add(...)
delete (...)
getInActive(..)
Get All Accounts that have not been active after the date: '12-31-99'
SELECT *
WHERE last_used < '12-31-99'

Repository Pattern

Persistence operations are Atomic



Accounts	Nominee Updated Dt	...	...
...	....	....	....
AX17266	12/12/2020	...	...

Nominee	Percentage	ID	...
Jane	30	100	....

Domain Services

A Domain Object that implements the Domain functionality (or concept) that may not be modeled (naturally) as a behavior in any domain entity or value object.

Domain Service is aware of the domain objects

Characteristics of Domain Services:

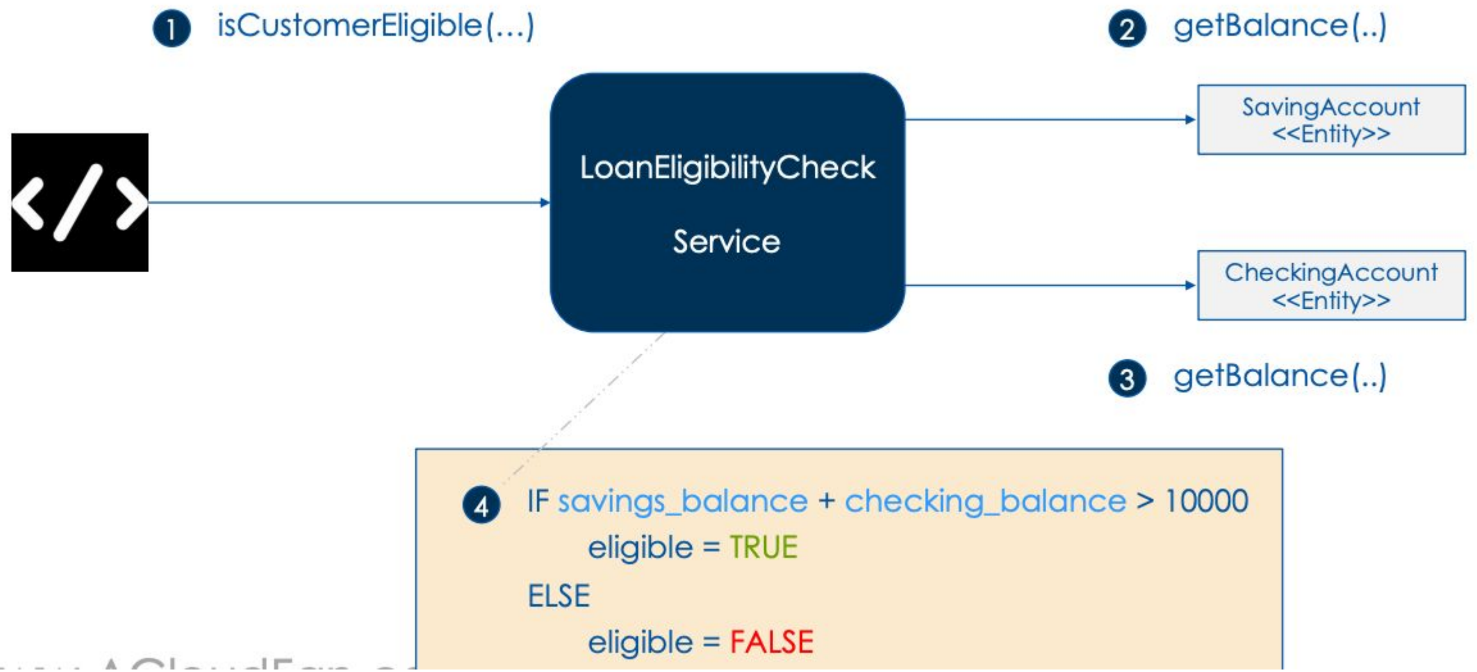
1- Business Behavior (i.e., Business Logic) for the Domain

2- Stateless

3- Highly cohesive

4- May interact with other Domain Services

Domain Services



```
4 IF savings_balance + checking_balance > 10000
      eligible = TRUE
  ELSE
      eligible = FALSE
```

Infrastructure Services

A service that interact with an external resource to address a concern that is not part of the primary problem domain.

Example infrastructure resources:

- Notifications, Email, SMS
- Logging system, Fluentd, ElasticSearch
- Persistence mechanism, Database, File system
- External APIs, Google maps, Salesforce API

Infrastructure Services

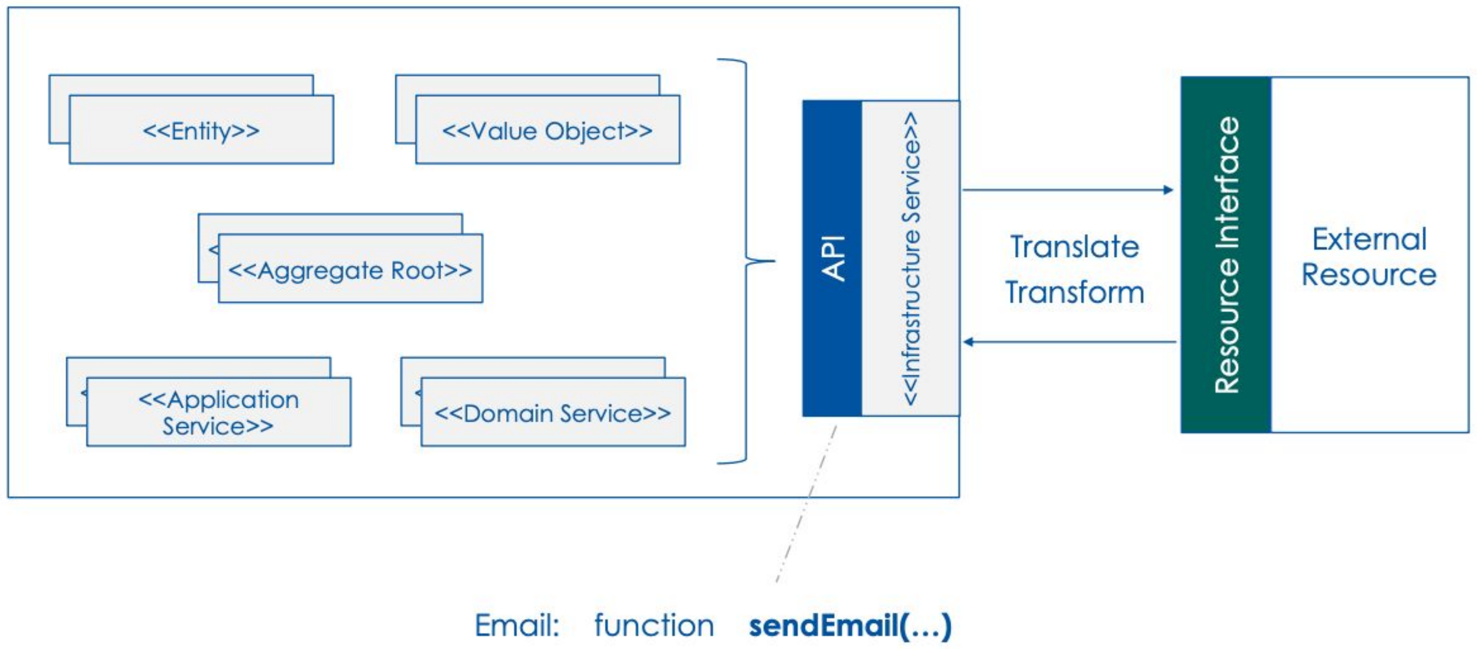
Characteristic of Infrastructure Service

1- NO Domain Logic

2- Single Responsibility

3- Standard Interface | Contract

Infrastructure Services



<<Entity>>
<<Value Object>>
<<Aggregate Root>>
<<Application Service>>
<<Domain Service>>
API
<<Infrastructure Service>>
Translate
Transform
Resource Interface
External Resource
Email: function sendEmail(...)