Domain Driven Design Quickly

Domain Driven Design

- collection of principles and patterns that help developers craft elegant object systems
- creates software abstractions called domain models
- models encapsulate complex business logic, closing the gap between business reality and code

DDD Main Concepts

- ➔ Ubiquitous Language
- ➔ Model-Driven Design
  - ◆ Layered Architecture
  - ◆ Entities
  - ◆ Value Objects
  - ◆ Services
  - ◆ Modules
  - ◆ Aggregates
  - ◆ Repositories
- ➔ Bounded Context

Ubiquitous Language

Ubiquitous Language

- is constructed during intensive communication between domain experts and developers
- consists of mainly words which are understandable both by domain experts and developers
- code should be written using the terms of this language

Ubiquitous Language

Roughly said:

- nouns become entities and value objects
- verbs become the methods

Layered Architecture

Layered Architecture

A common architectural solution for domain-driven designs contain four conceptual layers:

User Interface (Presentation Layer)	Responsible for presenting information to the user and interpreting user commands.
Application Layer	This is a thin layer which coordinates the application activity. It does not contain business logic. It does not hold the state of the business objects, but it can hold the state of an application task progress.
Domain Layer	This layer contains information about the domain. This is the heart of the business software. The state of business objects is held here. Persistence of the business objects and possibly their state is delegated to the infrastructure layer.
Infrastructure Layer	This layer acts as a supporting library for all the other layers. It provides communication between layers, implements persistence for business objects, contains supporting libraries for the user interface layer, etc.

Model-Driven Design

Entities

Entities

An object, that can be identified uniquely, by its identifier, which in code can be either ID, or combination of some attributes.

Example:

- bank account number can be identified by its account number uniquely
- airport has an international identifier used in travel agencies all over the world
- ...

Value Objects

Value Objects

are descriptors or properties important in the domain you are modeling

- ➔ do not have an identity
- ➔ they describe the things that do have identities
- ➔ should be immutable
- ➔ if you want a different value for the object, you simply create another one

Value Objects

***Example**:*

Customer -> customerID, name, street, city, state

can be represented as

Customer -> customerID, name, address (as an entity)

Address -> street, city, state (as a value object)

Difference between Entity & Value Object

1. Entity has an identity
2. Value objects mainly have no setters, they are immutable

Example 1:

Person - *is an entity*

Address - *is a value object, because 2 persons can have the same address*

Difference between Entity & Value Object

Example 2 - dollar:

a) In context of everyday life 10$ is a value object, because for you it doesn't matter if it's this 10$ or the other 10$.

b) In context of Government each individual $ makes difference, because each $ paper money has its own serialization number -> so in this context it's an entity.

How to determine if the object is an Entity OR Value Object ?

Ask the question:

Does it really matter which one?

If yes, then it's an entity.

Services

Services

Some verbs of Ubiquitous Language do not belong to any object, but they represent an important behavior of the domain.

- ➔ such behavior is declared as a service
- ➔ its purpose is to provide functionality for the domain

Modules

Modules

- ➔ Used as a method of organizing related concepts and tasks in order to reduce complexity. In enterprise application big model can be separated into modules.
- ➔ Modules should be made up of elements which functionally or logically belong together assuring cohesion.
- ➔ Modules should have well defined interfaces which are accessed by other modules. Instead of calling three objects of a module, it is better to access one interface, because it reduces coupling.
- ➔ Module names should be part of Ubiquitous Language.
- ➔ In Java they are simply packages.

Aggregates / Aggregate Root

Aggregates

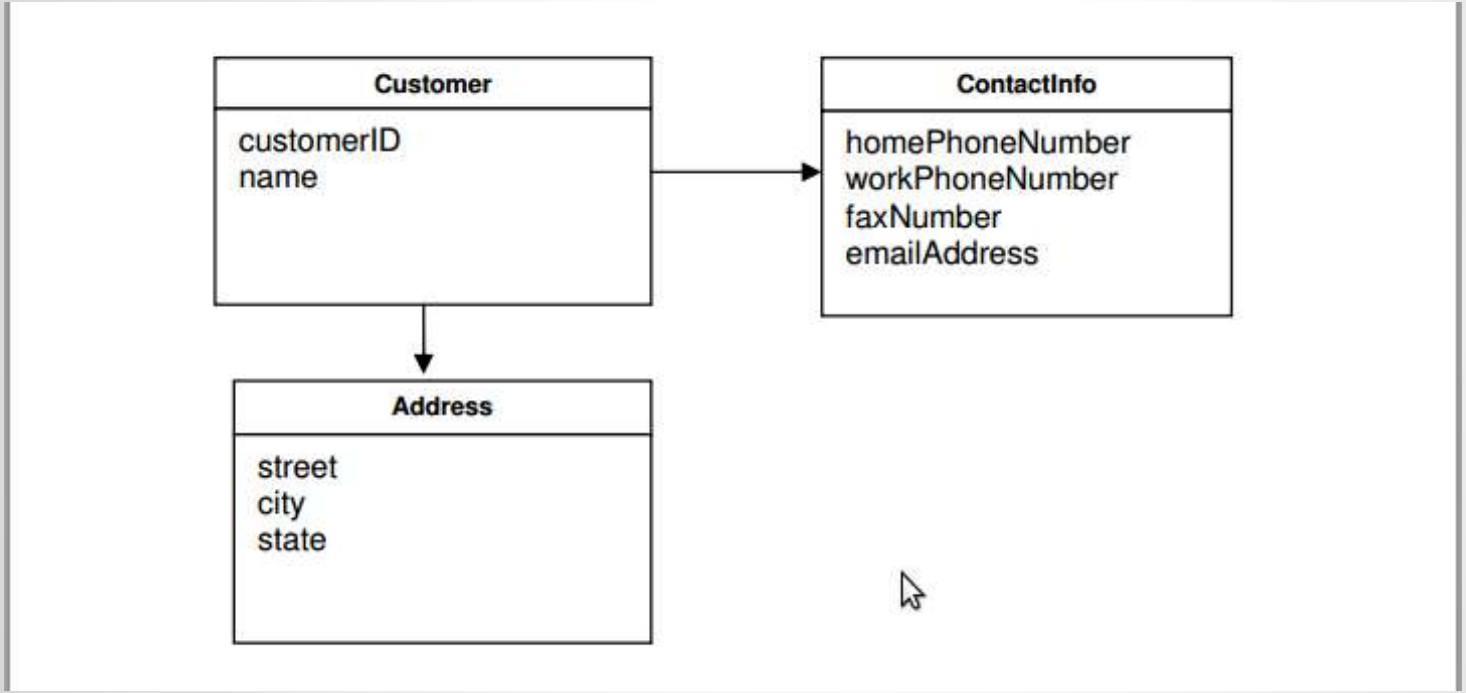
A group of associated objects which are considered as one unit with regard to data changes. It’s an entity which can contain other entities or value objects.

➔ has one Aggregate Root

➔ from outside only this root should be referenced/accessed

➔ only the root should be obtainable from DB through queries

Aggregates - Example

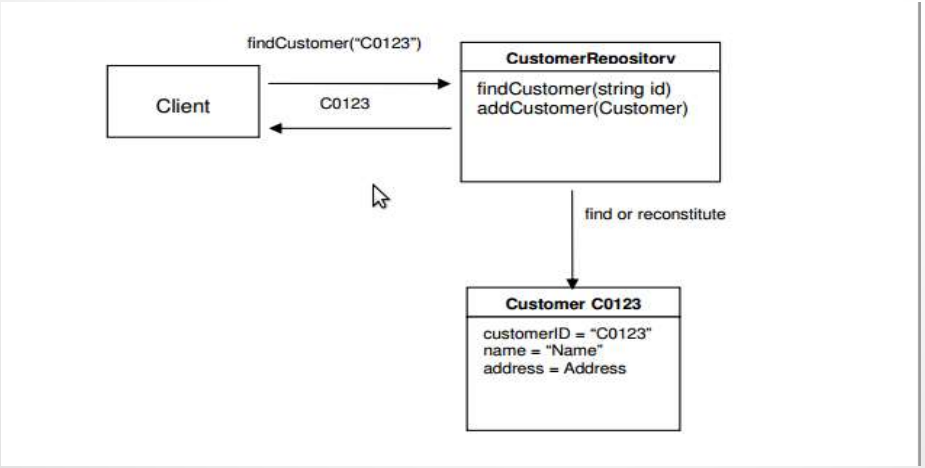


Repositories

Repositories

A set of methods used to retrieve objects.

The client calls method and passes one or more parameters which represent the selection criteria used to select an object or a set of matching objects.



Bounded Context

Bounded Context

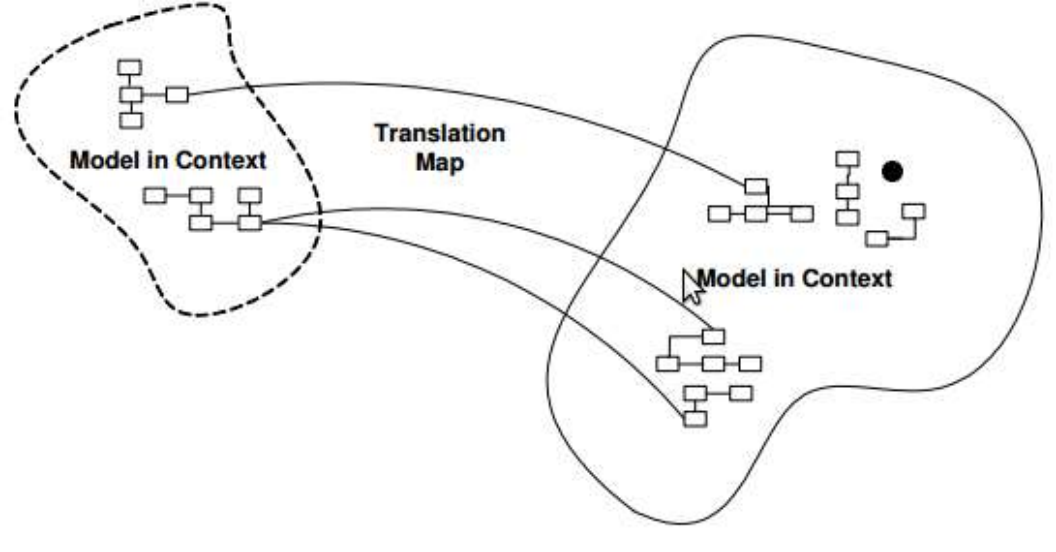
- ➔ an area of responsibility
- ➔ a logical frame inside of which the model evolves
- ➔ it owns its own Ubiquitous Language

- Use the context as the basis for team organization.
- The context of a model is the set of conditions which need to be applied to make sure that the terms used in the model have a specific meaning.
- Define the scope of a model, draw up the boundaries of its context.
- Explicitly set boundaries in terms of team organization, usage within specific parts of the application, and physical manifestations such as code bases and database schemas.
- Communicate via interfaces between different bounded context.

Context Map

Context Map

- ➔ a document which outlines the different Bounded Contexts and the relationships between them
- ➔ it can be a diagram like the one below, or it can be any written document.



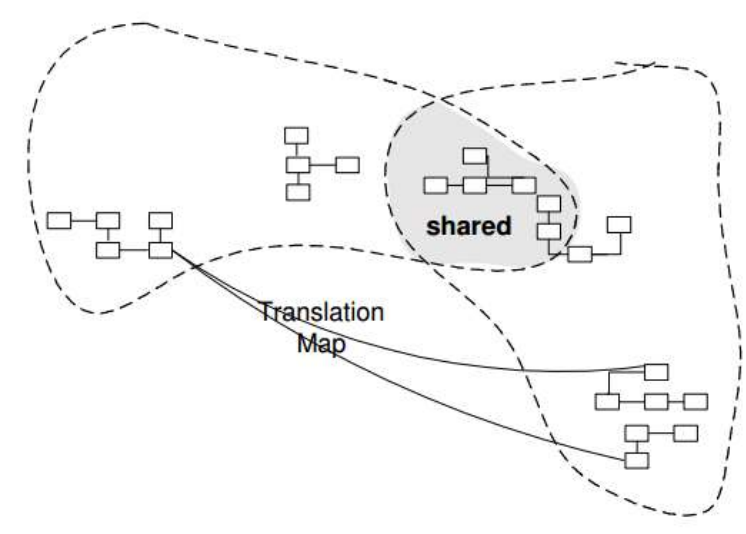
Context Map

There are series of patterns which can be used to create Context Maps where contexts have clear roles and their relationships are pointed out

- ➔ *Shared Kernel*
- ➔ *Customer-Supplier*
- ➔ *Separate Ways*
- ➔ *Open Host Services*
- ➔ *Anticorruption Layers*

Context Map - Shared Kernel

Shared domain model, which reduces duplication



Any change of the kernel should be communicated to another team, and the teams should be informed, making them aware of the new functionality.

Q & A