Domain Driven Design

(DDD)

Intro to DDD

- DDD is all about designing software on top of business domain concepts.
- Domain represents the subject area on which the application is being developed.
- So it is primarily focused on domain modeling and design aspects surrounding the domain.

Principles

Since the domain is the key it is required to construct the software design discipline centred on the principles that focus on

• core domain and understanding domain knowledge guided by domain experts

•agreeing on domain language that is to be used by stakeholders for the communication of domain business which is also called as Ubiquitous language

•improving the application model and resolving emerging domain-related issues by collaborating with domain experts

Must Haves

- Must work as one collaborative team
- No separation between domain experts (business experts, product owners) and developers (engineers, architects…)
- Iterative process to improve application model

Suitable For

- DDD is suitable for complex domains and large scale applications

- Separating problems into sub-domains (divide and conquer) so that problem of each sub-domain can be resolved independently

(with different sub-domains separation of concern is maintained)

Benefits of DDD

- DDD is flexible and scalable
- It is solution to customers perspective of the problem
- Well-organized and easily tested code
- Business logics are not scattered but lives in one place

Drawbacks

- Big learning curve for new principles, patterns and processes
- More time and effort to discuss and model the problem with domain experts
- Mostly suitable for complex domains

Building Blocks

In DDD there are number of high-level concepts which are objects of
DDD itself :

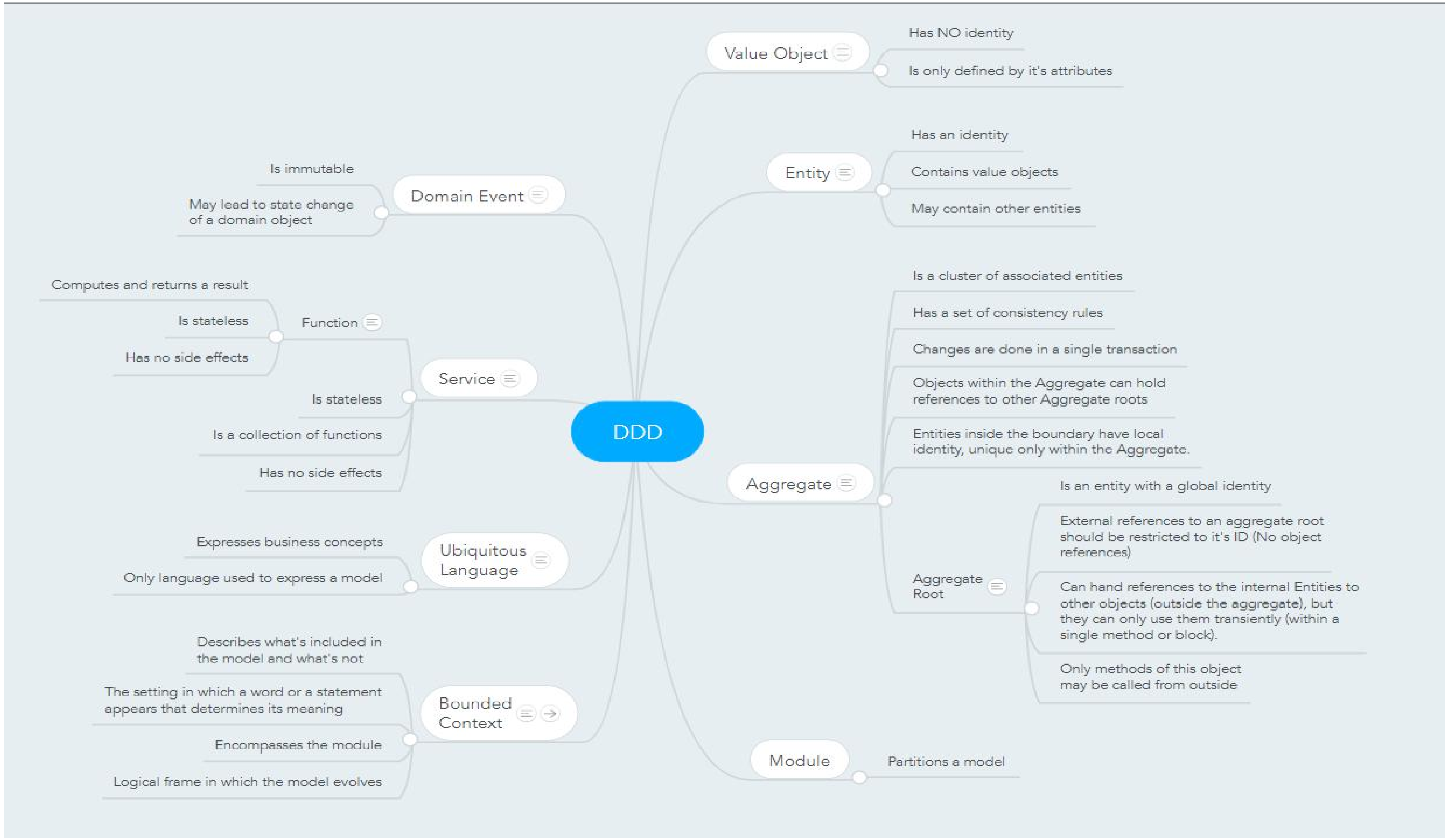
•Entity : These are the objects defined by identity and are mutable and uniquely identifiable by the identity we provide. Used to track, retrieve and store.

•Value Object: These are defined by value and are not mutable and
can be identified by composition of values, a common example is
string and date values. These measures , quantifies or describes a thing in the domain

- Domain Event : It is a event occured in the domain that you want other parts of the same domain (in-process) to be aware of. Created for event types which the domain experts care about
- Bounded-Context : A distinct part of the domain in which a particular subset of the ubiquitous language is consistent at all times. It is used for applying divide and conquer and splitting the domain model up into smaller models with clearly defined boundaries.
- Services : Provide a place in the model to hold behavior that doesn’t belong elsewhere in the domain
- Aggregrate : Is a composite of domain objects that can be treated as a single unit and are the basic element of transfer of data storage. Aggregrates are domain concepts like (order, clinic visit, playlist).

- Repositories : These are service that uses a global interface to provide access to all entities and value objects that are within a particular aggregate collection. Methods should be defined to allow for CRUD procees of objects within the aggregate.
- Factories : These are for creating complex objects and aggregates encapsulating the inner-workings of object manipulation

DDD Mind Map


DDD
Value Object
Has NO identity
Is only defined by it's attributes
Entity
Has an identity
Contains value objects
May contain other entities
Aggregate
Is a cluster of associated entities
Has a set of consistency rules
Changes are done in a single transaction
Objects within the Aggregate can hold references to other Aggregate roots
Entities inside the boundary have local identity, unique only within the Aggregate.
Aggregate Root
Is an entity with a global identity
External references to an aggregate root should be restricted to it's ID (No object references)
Can hand references to the internal Entities to other objects (outside the aggregate), but they can only use them transiently (within a single method or block).
Only methods of this object may be called from outside
Module
Partitions a model
Domain Event
Is immutable
May lead to state change of a domain object
Service
Function
Computes and returns a result
Is stateless
Has no side effects
Is stateless
Is a collection of functions
Has no side effects
Ubiquitous Language
Expresses business concepts
Only language used to express a model
Bounded Context
Describes what's included in the model and what's not
The setting in which a word or a statement appears that determines its meaning
Encompasses the module
Logical frame in which the model evolves

Thanks !