УЕБ-БАЗИРАНО ПРОГРАМИРАНЕ

Павел Кюркчиев
Ас. към ПУ „Паисий Хилендарски“
https://github.com/pkyurkchiev
@pkyurkchiev

DOMAIN-DRIVEN DESIGN (DDD)

Какво е домейн (domain)?

- Общото определение на домейн в речника е: „Сфера на знания или дейност“

От гледна точка на софтуерния разработчик:

- Домейнът се отнася до темата, в която приложението е предназначено да се прилага „сферата на знанието и дейността, около която се върти логиката на приложението“.

Определение на Domain-driven design

- Domain-driven design (DDD) е подход за разработка на софтуер за комплексни нужди чрез свързване и внедряването на развиващ се модел.

Базови принципи

- Съсредоточете се върху основната област и логиката на домейна.
- Базирайте сложните дизайни на модели от домейна.
- Постоянно си сътрудничете с експерти от домейн областта, за да подобрите модела на приложение и да разрешите всички възникващи проблеми, свързани с домейна.

Компоненти

Модел (Model)

- Система от абстракции, която описва избрани аспекти на даден домейн и може да се използва за решаване на проблеми, свързани с този домейн.

Контекст (Context)

- Настройката, в която се появява дума или израз, която определя нейното значение. Изявления за даден модел могат да бъдат разбрани само в определен контекст.

Вездесъщ език (Ubiquitous Language)

- Език, структуриран около модела на домейна и използван от всички членове на екипа за свързване и разбиране на всички дейности по разработката на софтуера.

Ограничен контекст (Bounded Context)

- Описание на граница (обикновено подсистема или работа на конкретен екип), в рамките на която се прилага или дефинира определен модел.

Изграждащи блокове:

- Entity
- Value Object
- Domain Event
- Aggregate
- Service
- Repositories
- Factories

Обект (Entity)

- Обект, който се отличава чрез идентификационен номер, за разлика от традиционните обекти, които се определят от техните атрибути. Ние можем да променим всичко в тези обекти (без неговия идентификационен номер) и той да запази.

Кола (Car)

Стойност обект (Value Object)

- Неизменяем (непроменим) обект, който има атрибути, но не и идентичност (идентификационен номер).

Гуми (Tires)

Домейн събитие (Domain Event)

- Обект, който се използва за запис на дискретно събитие, свързано с активност на модела в системата. Въпреки, че всички събития в системата могат да бъдат проследявани, събитие в домейна се създава само за типове събития, от които се интересуват експертите по домейна.

История на моторното масло (History of motor oil)

Съвкупност, агрегат (Aggregate)

- Клъстер от обекти и стойност обекти с дефинирани граници около групата. Вместо да позволява на всеки отделен обект или стойност обект да изпълнява всички действия самостоятелно, на колективната съвкупност от елементи се присвоява единичен обобщен корен елемент. Външните обекти нямат директен достъп до всеки отделен обект или стойност обект в съвкупността, а вместо това имат достъп само до единствения агрегиран корен елемент и го използват, за да предават инструкции на групата като цяло.

Машина (Machine)

Услуга (Service)

- По същество услугата е операция или форма на бизнес логика, която естествено не се вписва в сферата на обектите. С други думи, ако трябва да съществува някаква функционалност, но тя не може да бъде свързана с обект или стойност обект, това вероятно е услуга.

Създай кола (Create a car)

Хранилища (Repositories)

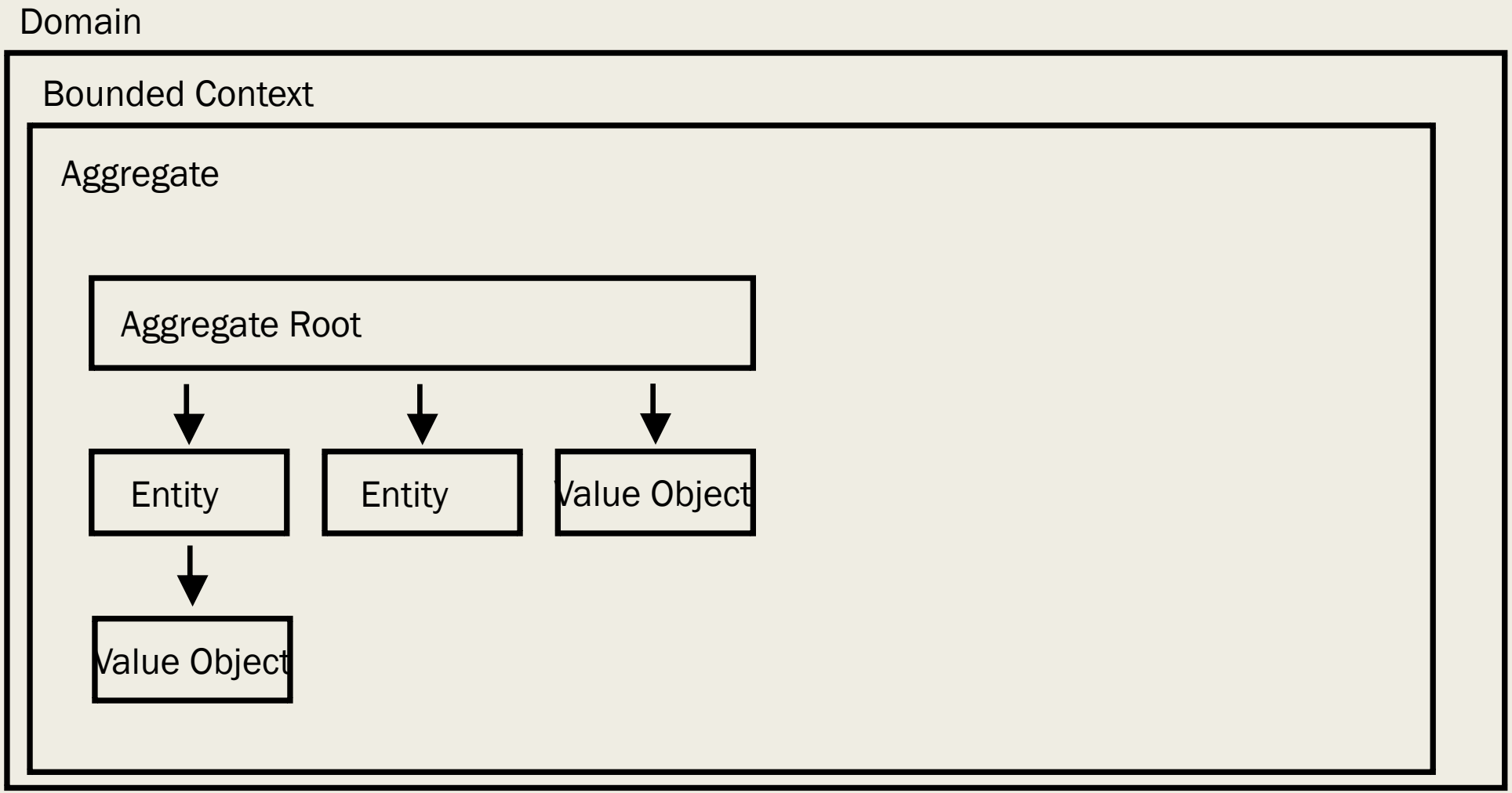
- Хранилищата нямат нищо общо с хранилищата в контрол на версиите (version of control, git, svn ...), значението на хранилище в DDD е услуга, която използва глобален интерфейс за осигуряване на достъп до всички обекти и стойност обекти, които са в определена съвкупна група.

- Определят методи, които да позволяват създаване, модификация и изтриване на обекти от съвкупността. Чрез използването на тази услуга хранилище за извършване на заявки за данни, целта е да се премахне възможността за записване на данни от бизнес логиката на обектните модели.

Фабрики (Factories)

- DDD предлага използването на фабрика, която капсулира логиката на създаване на сложни обекти и агрегати, като гарантира, че клиентът няма познания за вътрешната работа на манипулираните обекти.

Домейн


Domain
Bounded Context
Aggregate
Aggregate Root
Entity
Entity
Value Object
Value Object

Предимства на DDD:

- Улеснява комуникацията
- Подобрява гъвкавостта
- Подчертава домейн над интерфейс

Улеснява комуникацията

- С акцент върху установяването на общ и повсеместен език, свързан с домейн модела на проекта, екипите често намират комуникацията през целия жизнен цикъл на приложението много лесна. Обикновено DDD изисква по-малко технически жаргон, когато се обсъждат аспекти на приложението, тъй като общия език дефинира по-прости термини.

Подобрява гъвкавостта

- DDD е силно базиран около концепциите за обектно-ориентиран анализ и дизайн, почти всичко в рамките на домейн модела ще се основава на обект и следователно ще бъде доста модулно и капсулирано. Това позволява редовно и непрекъснато да се променят и подобряват различни компоненти.

Подчертава домейн над интерфейс

- DDD се изгражда около концепциите за домейн, и това което съветват експертите по домейна в рамките на проекта. DDD често ще произвежда приложения, които са точно подходящи и представителни за съответния домейн, за разлика от приложенията, които наблягат на UI/UX на първо място. Въпреки че е необходим очевиден баланс, фокусът върху домейна означава, че DDD подходът може да произведе продукт, който да отговаря най - добре на аудиторията, свързана с този домейн.

Недостатъци на DDD:

- Изисква стабилна експертиза на домейна
- Насърчава итеративните практики
- Неподходящ за сложни технически проекти

Изисква стабилна експертиза на домейна

- Дори и с най-опитните технически специалисти, работещи върху разработката, всичко може да се провали, ако в екипа няма поне един експерт по домейна, който да знае точните характеристики. В някои случаи DDD може да изисква интегрирането на един или повече външни членове към екипа, които да действат като експерти в областта на домейна през целия жизнен цикъл на приложението.

Насърчава итеративните практики

- Макар че мнозина биха считали това за предимство, не може да се отрече, че практиките на DDD силно разчитат на постоянна итерация и непрекъсната интеграция с цел изграждане на добър проект, който може да се коригира, когато е необходимо. Някои организации може да имат проблеми с тези практики, особено ако техният предишен опит е до голяма степен обвързан с по-малко гъвкави модели за развитие, като например модела на водопада или други подобни.

Неподходящ за сложни технически проекти

- DDD е чудесен за приложения, където има голяма степен на сложност на домейна (където бизнес логиката е доста сложна и объркана), DDD не е много подходящ за приложения, които имат не голяма сложност на домейна, но имат голяма техническа сложност. Проектът, който е технически сложен, може да бъде предизвикателство за разбиране от експертите в областта на домейните, което води до проблеми.

- Всичко това се случва, когато техническите изисквания или ограничения не са били напълно разбрани от всички членове на екипа.

DEMO BLAZING PIZZA

https://github.com/pkyurkchiev/web-programming-biel/tree/master/examples/BlazingPizza

ВЪПРОСИ ?