AGREGAT

Jednym z najważniejszych wzorców w technice DDD, jest agregat – hermetyczny graf obiektów. Są one główną jednostką logiki w modelowaniu DDD.

Agregatem jest klasa, która enkapsuluje wyspecjalizowana grupę reguł, związanych ze sobą biznesowo. Reguły tworzą razem jedna całość, mającą sens i są spójne natychmiast. Agregaty hermetyzują swoja strukturę – to znaczy, że nie udostępniają swoich szczegółów na zewnątrz. Jest to świetne rozwiązanie, mianowicie agregat, który jest zamknięty na świat zewnętrzny możemy w razie potrzeby modyfikować i wprowadzać potrzebne zmiany niczego nie psując w pozostałym kodzie. „Jest agentem, który odbiera sygnały. Czyli jest nastawiony na metody, a nie na struktury danych".

Charakteryzuję się tym, że reguły zawsze są po jego stronie. Może mieć w sobie inne obiekciki, chodzi o to że, klasa publiczna, która ma swoje API oraz metody - czyli korzeń agregatu, może wysyłać sygnały do innych obiektów, ale wyłącznie w obrębie siebie. Nigdy nie wychodzi na zewnątrz, (getter), chyba, że jest to uzasadnione. Wszystko odbywa się przy pomocy metod biznesowych. Biorąc pod uwagę przykład działa sztuki w galerii, to po stronie produktu (naszego dzieła) są reguły, kto, kiedy i na jakich prawach go zarezerwował.

Trzeba wystrzegać się tzw. „boskich klas" zawierających „wszystko" i odpowiadających „za wszystko". Agregat nie może być tzw. „workiem". Zbyt duże agregaty powodują niepowodzenia w modelowaniu opartym o DDD. Lepiej jest mieć małe agregaty, robiące pojedyncze rzeczy. Wtedy łatwo mogę wykorzystywać takie agregaty w różnych procesach. Nigdy nie może być tak, że agregat zawiera w sobie inny agregat. Może mieć w sobie nr innego agregatu.

Przykład: Wynajem samochodu w wypożyczalni.

Jednym z agregatem w wypożyczalni samochodów jest „CAR". To samochód wie czy jest dostępny do wypożyczenia, kto go wypożyczył, na jak długo.

POLICY - POLITYKI

Polityka w modelowaniu DDD to nic innego jak wzorzec strategii. Służy do wyznaczania zachować operacji biznesowych. Przykładową polityką może być naliczanie podatku przy zakupie produktu zależnie od kraju. Korzystając ze wzorca Policy otwieramy się na rozbudowę – czyli w monecie kiedy przyjdzie konieczność nowego sposobu przeliczenia podatku wystarczy dopisać nowy 'feature' i gotowe. Wszystkie polityki będą niezależne.

Przykład: Wynajem samochodu w wypożyczalni.

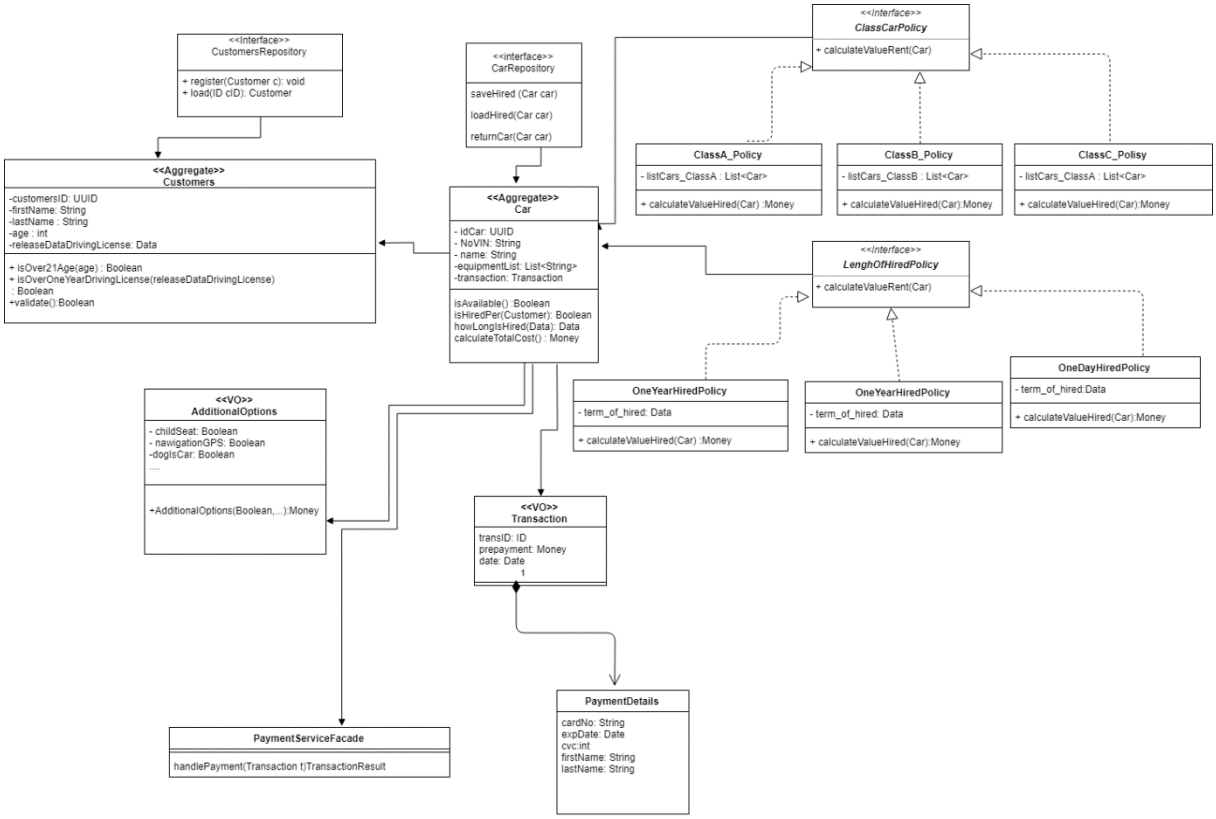
Mamy możliwość wypożyczenia samochodu na okres: {One_Day, One_Year,One_Month, ……}, każda polityka ma w sobie sposób przeliczania opłat za wypożyczenie samochodu zależnie od tego na jaki okres jest wypożyczony, np. im dłuższy okres wypożyczenia – tym mniejsza opłata w skali jednego dnia. Każdą dodatkową politykę z łatwością możemy dołączyć do całego systemu, nic nie zmieniając w całości naszego systemu.

REPOSITORY - REPOZYTORIUM

Repozytorium służy do zarządzania trwałością agregatu danego typu. W repozytorium nie umieszczamy wielu metod służących do wyciągania tabelek dla GUI. Odpowiada za wyszukiwanie – ale tylko obiektów potrzebnych do wykorzystania w metodach biznesowych.

Wykorzystywany jest do zapisu czy też pobrania agregatu po jego identyfikatorze. Za pomocą repozytoriów, wstrzykiwane są do agregatów obiekty współpracujące. Ponad to są abstrakcja nad magazynem danych (np. baza danych). Zwracają, a także odpowiadają za zapisywanie obiektów.

DIAGRAM MODELU



OPIS MODELU

1. Agregat „Car” – główny agregat.
    a. Metoada isAvailable – sprawdza czy samochód jest aktualnie dostępny do wypożyczenia.
    b. isHiredPer - Przez kogo jest wypożyczony.
    c. howLongIsHired – jak długo / do kiedy jest ważne wypożyczenie.
    d. calculateTotalCost – zwraca całkowity koszt wypożyczenia.
2. CarRepository – repozytorium do klasy Car.
    a. Odpowiada za obsługę obiektu Car w bazie danych.
3. Customer – sprawdza po rejestracji czy użytkownik, który wypożycza auto ma:
    a. isOver21Age – sprawdza na podstawie argumentu „age” czy ma więcej niż 21 lat.
    b. isOverOneYearDrivingLicense – sprawdza na podstawie argumentu „releaseDataDrivingLicense” czy prawo jazdy posiada dłużej niż jeden rok.

    c. Validate – wyznacza czy klient kwalifikuje się do wypożyczenia samochodu.
4. CustomerRepository – obsługuję rejestrację i odnalezienie użytkownika w bazie danych.
5. ClassCarPolicy – polityka klas/ segmentów samochodów. Na podstawie polityki mamy możliwość wyboru segmentu co wiąże się jakością samochodu, podstawowym wyposażeniem. – im lepsza klasa auta, tym lepsze wyposażenie podstawowe. – na konkretnym modelu jest tylko 3 klasy, jednak w rzeczywistości byłoby ich zdecydowanie więcej.
6. LenghOfHiredPolicy – polityka według której przeliczany jest koszt wynajęcia samochodu na podstawie długości wynajmu. Im dłuższy wynajem tym tańszy koszt wyposażenia. – na konkretnym modelu jest tylko 3 opcje, jednak w rzeczywistości byłoby ich zdecydowanie więcej.
7. AdditionalOptions – ValueObject posiadający dodatkowe opcje takie jak możliwość przewożenia psa w samochodzie, nawigacja, fotelik dziecięcy. Metoda zwraca całkowity koszt dodatkowych opcji.
8. Transaction – ValueObject posiadający ID opłacanej transakcji, datę, oraz kwotę przedpłaty wymaganej do realizacji wynajęcia samochodu.
9. Klasa PaymentDetails przechowuje dane dotyczące płatności.
10. PaymentServiceFacade – zewnętrzny system płatności.