도메인 주도 설계
6장. 도메인 객체의 생명주기

발표자: 김성중

도메인 객체 생명주기 이슈와 해결 방안

1. 생명주기 동안 무결성 유지하기
   - **집합체(AGGREGATE)**를 이용해 엄격한 모델 만들기
2. 생명주기 관리의 복잡도를 낮춰 난해한 모델이 되지 않도록 만들기
   - 생명주기의 초기단계의 책임을 위임하기
     -> **팩토리 (FACTORY)**
   - 생명주기의 중간/마지막단계(인프라스트럭처)의 책임을 위임하기
     -> **리파지토리 (REPOSITORY)**

집합체 (AGGREGATE)

엄격한 모델 만들기

집합체 (AGGREGATE)

- 목적
  - 모델 내에서 여러 객체로 구성된 한 객체의 범위를 어디서부터 어디까지 인지 정의하여 **참조에 대한 캡슐화**를 추상화 할 수 있다.
  - 모델 내 객체의 일관성을 보장하기 위해서는 개별 객체만이 아닌 서로 밀접한 관계에 있는 **객체 집합에도 불변식을 적용**할 수 있다.

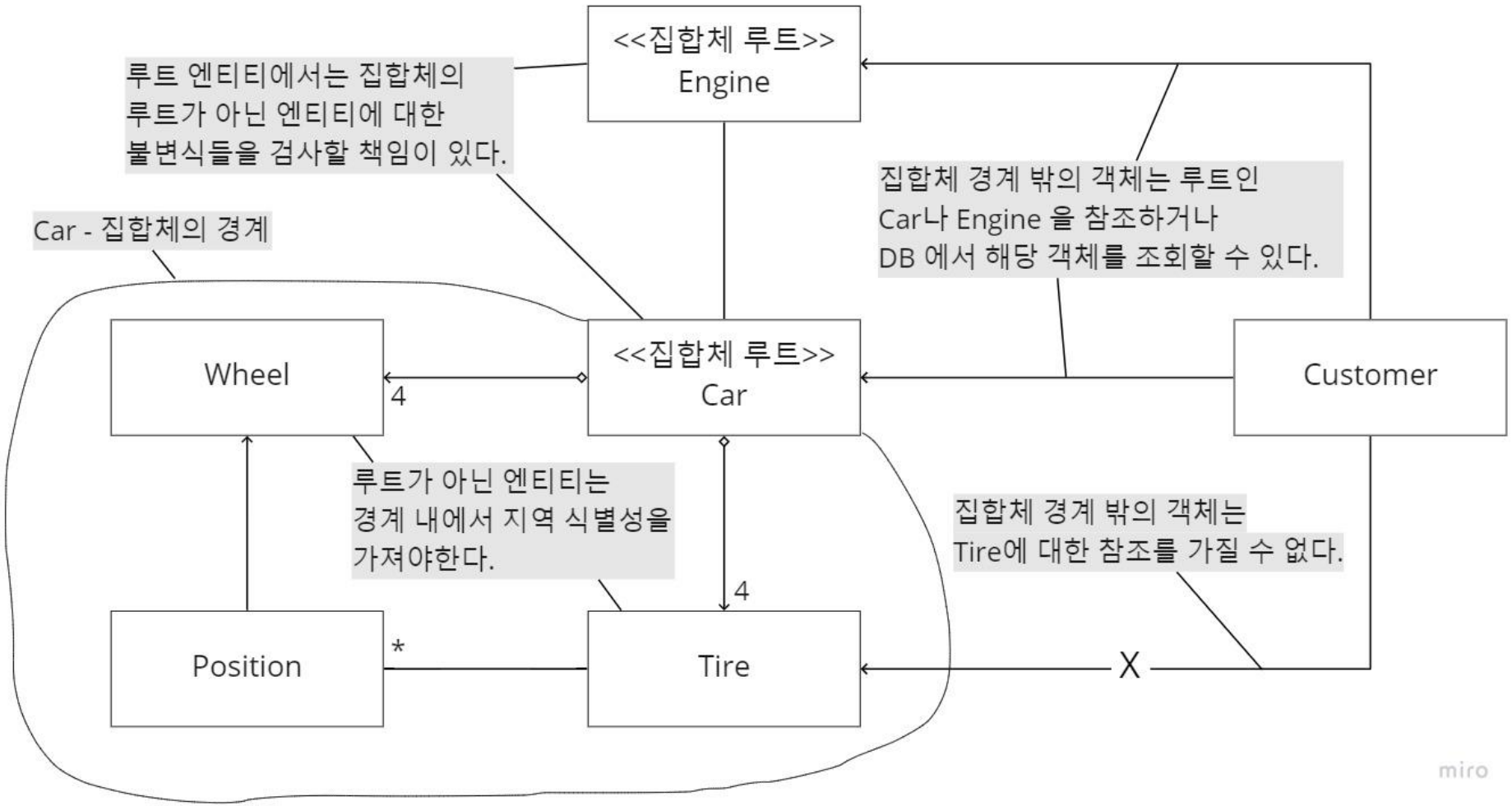
집합체의 구성

- 경계 (boundary)
  - 집합체에 무엇이 포함되고, 무엇이 포함되지 않는지를 정의한다.

- 루트 (root)
  - 집합체에 포함된 특정 엔티티를 가리킨다.
  - 하나의 집합체의 하나의 루트만 존재한다.
  - 경계 바깥에서는 루트만 참조할 수 있다.

- 루트 이외의 엔티티
  - 지역 식별성을 지닌다. (=집합체 내에서 식별성을 가진다.)

집합체 예시


<<집합체 루트>>
Engine
루트 엔티티에서는 집합체의
루트가 아닌 엔티티에 대한
불변식들을 검사할 책임이 있다.
집합체 경계 밖의 객체는 루트인
Car나 Engine 을 참조하거나
DB 에서 해당 객체를 조회할 수 있다.
Car - 집합체의 경계
Wheel
4
<<집합체 루트>>
Car
Customer
루트가 아닌 엔티티는
경계 내에서 지역 식별성을
가져야한다.
집합체 경계 밖의 객체는
Tire에 대한 참조를 가질 수 없다.
4
Position
*
Tire
X
miro

집합체에 적용해야 할 규칙

루트 엔티티

- 전역 식별성을 지닌다.
- 불변식을 검사할 책임이 있다.
  - 경계내의 어떤 변경에도 불변식을 지켜야한다.
- 경계 밖에서 참조할 수 있다.
- 질의를 통해 획득할 수 있다.
- 삭제 시 경계 안 모든 요소를 삭제한다.

경계 안의 엔티티

- 지역 식별성을 지닌다.
- 경계 밖에서 참조를 제한한다.
  - 일시적 참조 or VO 제공
- 루트 엔티티로부터 탐색된다.
- 다른 집합체의 루트를 참조할 수 있다.

집합체의 효과

- 생명주기의 전 단계에서 **불변식이 유지돼야 할 범위**를 표시해 준다.

- **루트 엔티티 접근**으로 **집합체 전체의 불변식**을 효과적으로 이행할 수 있다.

팩토리 (FACTORY)

생명주기 초기, 생성의 책임

팩토리 (FACTORY)

- 필요성
  - 복잡한 조립 연산에 대한 책임
    - 객체 – “어떻게 만들어졌나 " 라는 책임은 객체에게 어울리지 않는다.
    - 클라이언트 – 캡슐화를 위반 / 설계의 복잡성 / 지나친 결합도의 위험성이 있다.

- 팩토리
  - 다른 **객체를 생성하는 책임**을 가지는 프로그램 요소
  - 복잡한 객체 및 집합체를 **생성하는 지식을 캡슐화** 한다.
  - 객체의 **생성과 재구성**이라는 **생명주기 전이를 캡슐화**한다.

팩토리 설계 (1)

팩토리 설계의 기본 요건

1. 생성 방법은 원자적(atomic)이어야 한다.
2. 적절하지 않은 생성에 대한 예외처리가 수반되어야 한다.
3. 생성된 객체나 집합체는 불변식을 모두 지켜야한다.
4. 반환 타입은 생성된 클래스가 아닌 생성하고자 하는 타입으로 추상화돼야 한다.

생성자만으로 충분한 경우

- 다형적이거나 계층구조의 일부를 구성하지 않는 타입 클래스인 경우
- 클라이언트가 정책 선택을 위해 구현체 자체에 관심이 있는 경우
- 클라이언트가 객체의 속성을 이용할 수 있고 공개된 생성자에서 중첩 생성되지 않는 경우
- 생성자가 복잡하지 않은 경우
- 공개 생성자가 FACTORY 와 동일한 원자적 규칙을 반드시 준수해야하는 경우

팩토리 설계 (2)

- 팩토리와 팩토리의 위치선정

1. 특정 객체가 팩토리 메서드를 가지는 경우
   - 집합체 루트에서 루트가 아닌 엔티티를 생성하는 경우 (소유의 관계)
     - 예) 구매 주문에 주문할 품목을 추가하는 경우
   - 다른 객체를 만들어내는 것과 관련 있는 객체 (생성해 주는 관계)
     - 예) 중개업자가 새로운 거래 주문을 중개해주는 경우
   - 위와 같은 경우는 생성물과 가장 밀접한 관계의 객체에 팩토리가 있어야한다.
2. 독립형 팩토리를 만드는 경우
   - 구상하는 구현체나 복잡한 생성과정을 감춰야 하는 경우
   - 집합체를 생성하여 루트의 참조를 건내야하는 경우
     - 이 경우에 불변식에 대한 책임을 팩토리가 가질 수 있다.

팩토리 설계 (3)

- 인터페이스 설계
  - **각 연산은 원자적이어야 한다**. 생성을 위한 모든 정보를 한 번에 팩토리로 전달해야 한다. 또한, 실패에 대한 표준을 명세할 필요가 있다.
  - **팩토리는 자신에게 전달된 인자와 결합**될 것이다. 의존성의 덫에 빠지지 않도록 주의해야한다.
- 불변식 로직을 팩토리에 위임하는 경우
  - 식별속성 할당과 관련 있거나 생성물이 VO 인 경우(일회성이기 때문에)
  - 생성물을 단순하게 유지하려는 경우
- 엔티티 팩토리와 VO 팩토리
  - VO 는 불변하므로 모든 정보를 한 번에 생성해야 한다.
  - 엔티티는 유효한 집합체를 만들기위한 필수 속성만 받아들이는 경향이 있다.
  - 엔티티의 식별 속성 할당의 책임을 가지기에 적절한 곳이다.

재구성을 위한 팩토리

- 생성을 위해 사용하는 팩토리와의 주요 차이점
  - 식별 속성을 할당하지 않는다.
  - 불변식 위반에 대한 예외처리를 다른 방식으로 처리한다.

리파지토리 (REPOSITORY)

생명주기 중간/마지막 단계 책임

리파지토리 (REPOSITORY)

- 리파지토리 역할
  - 실제 데이터 저장소에서 동작하는 객체 추가/제거 메서드를 제공하고, 연산을 캡슐화 한다.
  - 특정한 기준에 따라 클래스를 선택하고, 속성값을 만족하는 객체 및 객체 컬렉션을 반환하는 메서드를 제공하여 질의 기술을 캡슐화 한다.
  - 집합체의 경우 집합체 루트에 대해서만 리파지토리를 제공한다.
  - 모든 객체 저장과 접근은 리파지토리로 위임하고 모델에 집중한다.

리파지토리의 이점

- 리파지토리의 이점
  - 클라이언트에게 객체를 획득하거나 객체의 생명주기 관리를 위한 단순한 모델을 제시한다.
  - 영속화 기술, 데이터베이스 전략, 데이터베이스 소스에 대한 관심사를 애플리케이션과 도메인 설계로부터 분리해준다.
  - 객체를 접근하기 위한 기준을 설계할 수 있게 해준다.
  - 테스트에서 가짜데이터를 사용하기 쉬워진다.

리포지토리 설계 (1)

리포지토리에 질의하기

- 구체적인 매개 변수를 통한 질의
  - 식별자를 통한 질의
  - 특정 속성에 대한 연산을 토대로 질의
  - 값 범위에 대한 객체선택
  - 리포지토리 책임의 연산 수행 (ex. Avg, Max..)
- 명세에 기반을 둔 질의
  - 일반화된 질의 방법
  - 제 3의 객체를 이용해 원하는 조건을 선언

리포지토리 구현

- 저장, 조회 질의 메커니즘 캡슐화
  - 가장 기본적인 기능
- 타입을 추상화 한다.
  - 반드시 반환 타입이 아닌 추상 타입을 지양
- 클라이언트와의 분리를 활용한다.
  - 리파지토리 변경의 자유도 보장
- 트랜잭션 제어를 클라이언트에 둔다.
  - 트랜잭션 관리의 단순화

리포지토리 설계 (2)

클라이언트와 개발자

- 클라이언트 코드는 리파지토리의 구현을 무시한다.
  - 리파지토리는 구현이 분리되어 있다.
  - 리파지토리는 캡슐화가 잘 되어있다.

- 개발자는 리파지토리의 구현의 상세 과정을 이해한다.
  - 의도대로 동작하는지 관찰해야 한다.
  - 수행 성능에 문제가 있는지 관찰해야한다.

관계형 DB 를 위한 객체 설계

- "한 객체당 한 테이블 매핑"에 얽매이지 않는다.

- 객체 관계의 풍부함을 희생해 모델을 밀접하게 하거나, 정규화 같은 관계표준을 절충하여 객체 매핑을 단순화 한다.

- 객체 시스템 외부 프로세스에서 객체 저장소에 접근해서는 안된다.
  - 불변식을 위반할 수 있음
  - 데이터 모델 고착화로 리팩터링이 어려워질 수 있음

- 테이블의 외래키는 또 다른 엔티티 객체에 대한 참조로 변환해야 한다

팩토리와의 관계

팩토리

- 생명주기 초기를 다룸
- 기술적 관점
  - 새로운 객체를 생성
- 책임의 관점
  - 새로운 객체를 생성

리파지토리

- 생명주기 중간/마지막을 다룸
- 기술적 관점 차이
  - 기존 객체 정보를 찾아 객체를 생성
    - 본 관점에서는 리파지토리는 팩토리로 생각할 수 있다.
- 책임의 관점
  - 기존 객체를 찾음
    - 최초에는 생성하더라도, 그 이후는 다를 수 있다.
    - 최초의 동작의 일부는 팩토리에 위임될 수 있다.

- 팩토리로부터 영속화에 대한 모든 책임을 빼내고 리파지토리로 분리
  - 리파지토리에서는 일부 기능을 팩토리로 위임할 수 있음

끝

감사합니다