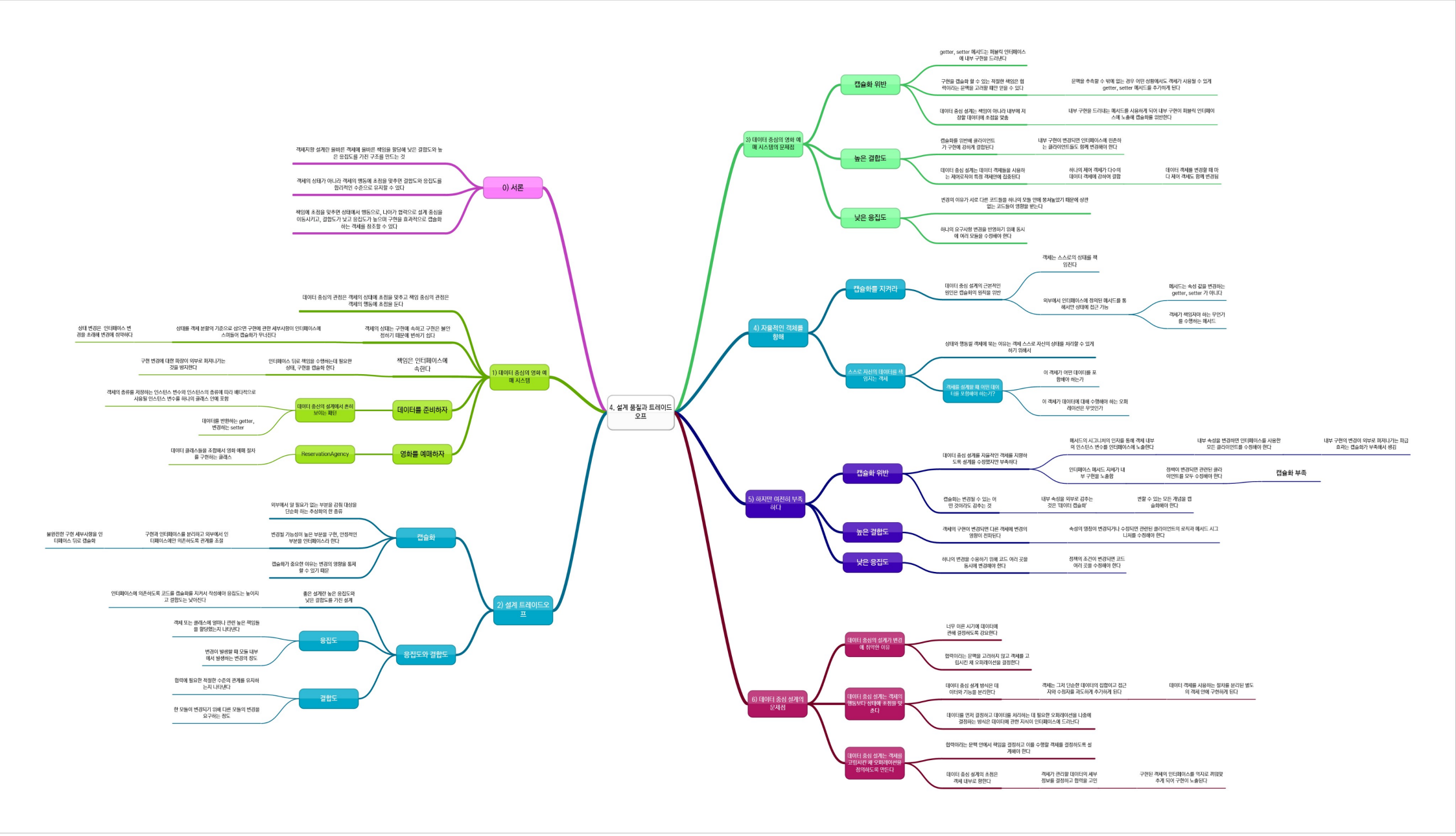
4. 설계 품질과 트레이드오프
0) 서론
객체지향 설계란 올바른 객체에 올바른 책임을 할당해 낮은 결합도와 높은 응집도를 가진 구조를 만드는 것
객체의 상태가 아니라 객체의 행동에 초점을 맞추면 결합도와 응집도를 합리적인 수준으로 유지할 수 있다
책임에 초점을 맞추면 상태에서 행동으로, 나아가 협력으로 설계 중심을 이동시키고, 결합도가 낮고 응집도가 높으며 구현을 효과적으로 캡슐화하는 객체를 창조할 수 있다
1) 데이터 중심의 영화 예매 시스템
데이터 중심의 관점은 객체의 상태에 초점을 맞추고 책임 중심의 관점은 객체의 행동에 초점을 둔다
객체의 상태는 구현에 속하고 구현은 불안정하기 때문에 변하기 쉽다
상태를 객체 분할의 기준으로 삼으면 구현에 관한 세부사항이 인터페이스에 스며들어 캡슐화가 무너진다
상태 변경은 인터페이스 변경을 초래해 변화에 취약하다
책임은 인터페이스에 속한다
인터페이스 뒤로 책임을 수행하는데 필요한 상태, 구현을 캡슐화 한다
구현 변경에 대한 파장이 외부로 퍼져나가는 것을 방지한다
데이터를 준비하자
데이터 중심의 설계에서 흔히 보이는 패턴
객체의 종류를 저장하는 인스턴스 변수와 인스턴스의 종류에 따라 베타적으로 사용될 인스턴스 변수를 하나의 클래스 안에 포함
데이터를 반환하는 getter, 변경하는 setter
영화를 예매하자
ReservationAgency
데이터 클래스들을 조합해서 영화 예매 절차를 구현하는 클래스
2) 설계 트레이드오프
캡슐화
외부에서 알 필요가 없는 부분을 감춰 대상을 단순화 하는 추상화의 한 종류
변경될 가능성이 높은 부분을 구현, 안정적인 부분을 인터페이스로 만든다
캡슐화가 중요한 이유는 변경의 영향을 통제할 수 있기 때문
구현과 인터페이스를 분리해고 외부에서 인터페이스에만 의존하도록 관계를 조절
불완전한 구현 세부사항을 인터페이스 뒤로 캡슐화
응집도와 결합도
좋은 설계란 높은 응집도와 낮은 결합도를 가진 설계
응집도
객체 또는 클래스에 얼마나 관련 높은 책임들을 할당했는지 나타낸다
변경이 발생할 때 모듈 내부에서 발생하는 변경의 정도
결합도
협력에 필요한 적절한 수준의 관계를 유지하는지 나타낸다
한 모듈이 변경되기 위해 다른 모듈의 변경을 요구하는 정도
인터페이스의 의존하도록 코드를 캡슐화를 지켜서 작성해야 응집도는 높이지고 결합도는 낮아진다
3) 데이터 중심의 영화 예매 시스템의 문제점
캡슐화 위반
getter, setter 메서드는 퍼블릭 인터페이스에 내부 구현을 드러낸다
구현을 캡슐화 할 수 있는 적절한 책임은 협력이라는 문맥을 고려할 때만 얻을 수 있다
문맥을 추측할 수 밖에 없는 경우 어떤 상황에서도 객체가 사용될 수 있게 getter, setter 메서드를 추가하게 된다
데이터 중심 설계는 책임이 아니라 내부에 저장할 데이터에 초점을 맞춤
내부 구현을 드러내는 메서드를 사용하게 되어 내부 구현이 퍼블릭 인터페이스에 노출돼 캡슐화를 위반한다
높은 결합도
캡슐화를 위반해 클라이언트가 구현에 강하게 결합된다
내부 구현이 변경되면 인터페이스에 의존하는 클라이언트들도 함께 변경해야 한다
데이터 중심 설계는 데이터 객체들을 사용하는 제어로직이 특정 객체안에 집중된다
하나의 제어 객체가 다수의 데이터 객체에 강하여 결합
데이터 객체를 변경할 때 마다 제어 객체도 함께 변경됨
낮은 응집도
변경의 이유가 서로 다른 코드들을 하나의 모듈 안에 뭉쳐놓았기 때문에 상관없는 코드들이 영향을 받는다
하나의 요구사항 변경을 반영하기 위해 동시에 여러 모듈을 수정해야 한다
4) 자율적인 객체를 위해
캡슐화를 지켜라
데이터 중심 설계의 근본적인 원인은 캡슐화의 원칙을 위반
객체는 스스로의 상태를 책임진다
외부에서 인터페이스에 정의된 메서드를 통해서만 상태에 접근 가능
메서드는 속성 값을 변경하는 getter, setter 가 아니다
객체가 책임지어야 하는 무언가를 수행하는 메서드
스스로 자신의 데이터를 책임지는 객체
상태와 행동을 객체에 묶는 이유는 객체 스스로 자신의 상태를 처리할 수 있게 하기 위해서
객체를 설계할 때 어떤 데이터를 포함해야 하는가?
이 객체가 어떤 데이터를 포함해야 하는가
이 객체가 데이터에 대해 수행해야 하는 오퍼레이션은 무엇인가
5) 하지만 여전히 부족하다
캡슐화 위반
데이터 중심 설계를 자율적인 객체를 지향하도록 설계를 수정했지만 부족하다
메서드의 시그니처의 인지를 통해 객체 내부의 인스턴스 변수를 인터페이스에 노출한다
내부 속성을 변경하면 인터페이스를 사용한 모든 클라이언트를 수정해야 한다
내부 구현의 변경이 외부로 퍼져나가는 파급효과는 캡슐화가 부족해서 생김
인터페이스 메서드 자체가 내부 구현을 노출함
절차이 변경되면 관련된 클라이언트를 모두 수정해야 한다
캡슐화 부족
캡슐화는 변경될 수 있는 어떤 것이라도 감추는 것
내부 속성을 외부로 감추는 것은 '데이터 캡슐화'
변할 수 있는 모든 개념을 캡슐화해야 한다
높은 결합도
객체의 구현이 변경되면 다른 객체에 변경의 영향이 전파된다
속성의 영향이 변경되거나 수정되면 관련된 클라이언트의 로직과 메서드 시그니처를 수정해야 한다
낮은 응집도
하나의 변경을 수용하기 위해 코드 여러 곳을 동시에 변경해야 한다
정책의 조건이 변경되면 코드 여러 곳을 수정해야 한다
6) 데이터 중심 설계의 문제점
데이터 중심의 설계가 변경에 취약한 이유
너무 이른 시기에 데이터에 관해 결정하도록 강요한다
협력이라는 문맥을 고려하지 않고 객체를 고립시킨 채 오퍼레이션을 결정한다
데이터 중심 설계는 객체의 행동보다 상태에 초점을 맞춘다
데이터 중심 설계 방식은 데이터와 기능을 분리한다
객체는 그저 단순한 데이터의 집합이고 접근자와 수정자를 과도하게 추가하게 된다
데이터 객체를 사용하는 절차를 분리된 별도의 객체 안에 구현하게 된다
데이터를 먼저 결정하고 데이터를 처리하는 데 필요한 오퍼레이션을 나중에 결정하는 방식은 데이터에 관한 지식이 인터페이스에 드러난다
데이터 중심 설계는 객체를 고립시킨 채 오퍼레이션을 정의하도록 만든다
협력이라는 문맥 안에서 책임을 결정하고 이를 수행할 객체를 결정하도록 설계해야 한다
데이터 중심 설계의 초점은 객체 내부로 향한다
객체가 관리할 데이터의 세부 정보를 결정하고 협력을 고민
구현된 객체의 인터페이스를 억지로 끼워맞추게 되어 구현이 노출된다