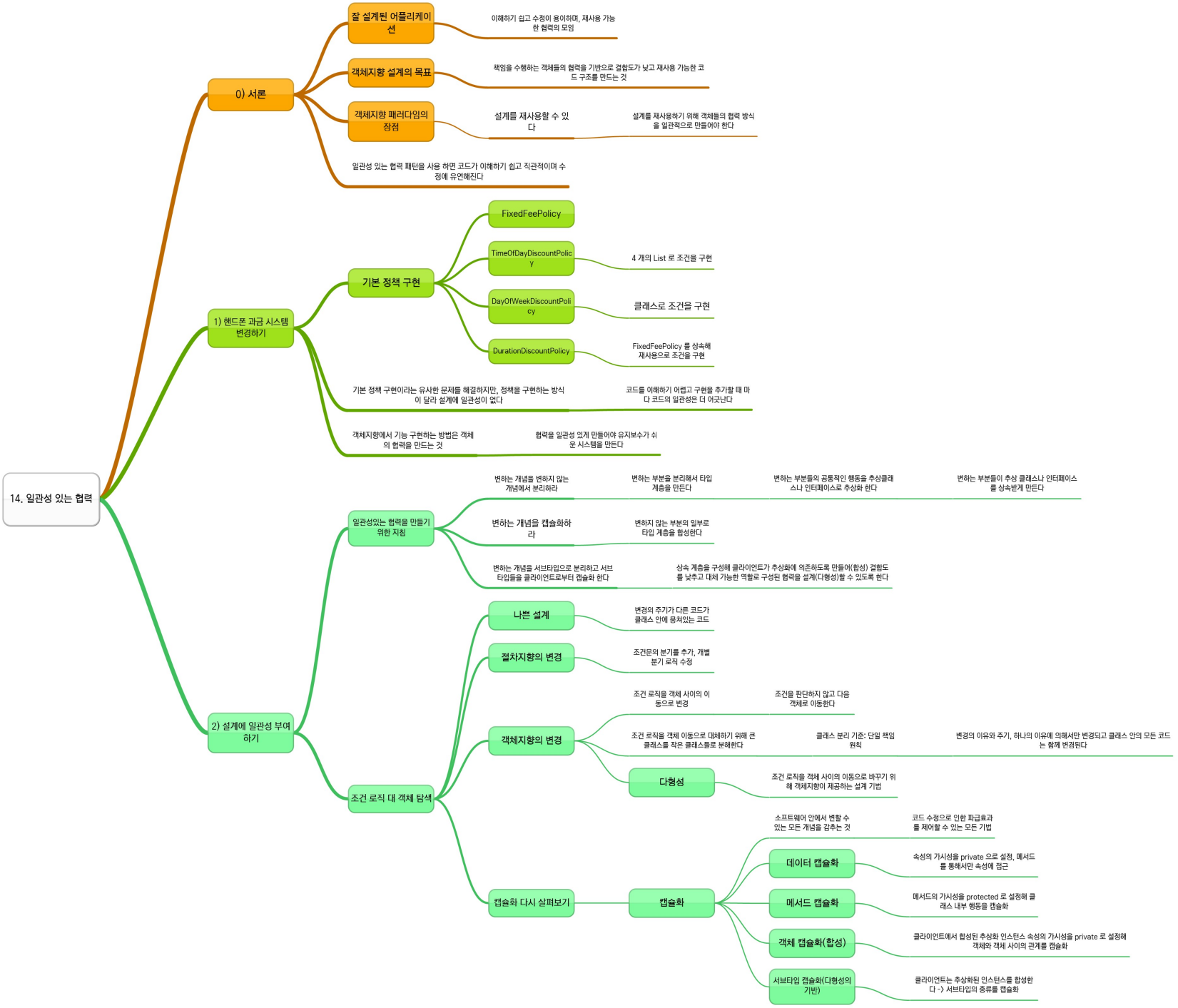
14. 일관성 있는 협력
0) 서론
잘 설계된 어플리케이션
이해하기 쉽고 수정이 용이하며, 재사용 가능한 협력의 모임
객체지향 설계의 목표
책임을 수행하는 객체들의 협력을 기반으로 결합도가 낮고 재사용 가능한 코드 구조를 만드는 것
객체지향 패러다임의 장점
설계를 재사용할 수 있다
설계를 재사용하기 위해 객체들의 협력 방식을 일관적으로 만들어야 한다
일관성 있는 협력 패턴을 사용 하면 코드가 이해하기 쉽고 직관적이며 수정에 유연해진다
1) 핸드폰 과금 시스템 변경하기
기본 정책 구현
FixedFeePolicy
TimeOfDayDiscountPolicy
4 개의 List 로 조건을 구현
DayOfWeekDiscountPolicy
클래스로 조건을 구현
DurationDiscountPolicy
FixedFeePolicy 를 상속해 재사용으로 조건을 구현
기본 정책 구현이라는 유사한 문제를 해결하지만, 정책을 구현하는 방식이 달라 설계에 일관성이 없다
코드를 이해하기 어렵고 구현을 추가할 때 마다 코드의 일관성은 더 어긋난다
객체지향에서 기능 구현하는 방법은 객체의 협력을 만드는 것
협력을 일관성 있게 만들어야 유지보수가 쉬운 시스템을 만든다
2) 설계에 일관성 부여하기
일관성있는 협력을 만들기 위한 지침
변하는 개념을 변하지 않는 개념에서 분리하라
변하는 부분을 분리해서 타입 계층을 만든다
변하는 부분들의 공통적인 행동을 추상클래스나 인터페이스로 추상화 한다
변하는 부분들이 추상 클래스나 인터페이스를 상속받게 만든다
변하는 개념을 캡슐화하라
변하지 않는 부분의 일부로 타입 계층을 합성한다
변하는 개념을 서브타입으로 분리하고 서브타입들을 클라이언트로부터 캡슐화 한다
상속 계층을 구성해 클라이언트가 추상화에 의존하도록 만들어(합성) 결합도를 낮추고 대체 가능한 역할로 구성된 협력을 설계(다형성)할 수 있도록 한다
조건 로직 대 객체 탐색
나쁜 설계
변경의 주기가 다른 코드가 클래스 안에 뭉쳐있는 코드
절차지향의 변경
조건문의 분기를 추가, 개별 분기 로직 수정
객체지향의 변경
조건 로직을 객체 사이의 이동으로 변경
조건을 판단하지 않고 다음 객체로 이동한다
조건 로직을 객체 이동으로 대체하기 위해 큰 클래스를 작은 클래스들로 분해한다
클래스 분리 기준: 단일 책임 원칙
변경의 이유와 주기, 하나의 이유에 의해서만 변경되고 클래스 안의 모든 코드는 함께 변경된다
다형성
조건 로직을 객체 사이의 이동으로 바꾸기 위해 객체지향이 제공하는 설계 기법
캡슐화 다시 살펴보기
캡슐화
소프트웨어 안에서 변할 수 있는 모든 개념을 감추는 것
코드 수정으로 인한 파급효과를 제어할 수 있는 모든 기법
데이터 캡슐화
속성의 가시성을 private 으로 설정, 메서드를 통해서만 속성에 접근
메서드 캡슐화
메서드의 가시성을 protected 로 설정해 클래스 내부 행동을 캡슐화
객체 캡슐화(합성)
클라이언트에서 합성된 추상화 인스턴스 속성의 가시성을 private 로 설정해 객체와 객체 사이의 관계를 캡슐화
서브타입 캡슐화(다형성의 기반)
클라이언트는 추상화된 인스턴스를 합성한 다 -) 서브타입의 종류를 캡슐화