* 11장 시스템
* 시스템 제작과 시스템 사용을 분리하라
  + 제작과 사용은 아주 다르다는 사실을 명심한다
  + 소프트웨어 시스템은 준비과정과 런타임 로직을 분리해야 한다.
  + 시작단계는 모든 애플리케이션이 풀어야 할 관심사.
  + 체계적이고 탄탄한 시스템을 만들고 싶다면 흔히 쓰는 좀스럽고 손쉬운 기법으로 모듈성을 깨서는 절대로 안된다. 객체를 생성하거나 의존성을 연걸할 때도 마찬가지다. 설정 논리는 일반 실행논리와 분리해야 모듈성이 높아진다. 또한 주요 의존성을 해결하기 위한 방식, 즉 전반적이며 일반적인 방식도 필요하다.
  + main 분리
    - 시스템 생성과 시스템 사용을 분리하는 한가지 방법
    - 생성과 관련된 코드는 모두 main이나 main이 호출하는 모듈로 옮기고 나머지 시스템은 모든 객체가 생성되었고 모든 의존성이 연결되었다고 가정한다.
    - Main이 구축자에 의해 구축되고 설정된 객체로 의해 제작이되고 애플리케이션이 구축자에 의해 생성이 된다.
    - 제어흐름은 따리가기 쉬운데 main함수에서 시스템에 필요한 객체를 생성한 후 이를 애플리케이션에 넘긴다.
  + 팩토리
    - 객체가 생성되는 시점을 애플리케이션이 결정할 필요도 생긴다.
  + 의존성 주입
    - 사용과 제작을 분리하는 매커니즘인 의존성 주입이 있다.
    - 의존성 주입은 제어 역전 기법과 의존성 관리에 적용한 메커니즘이다. 의존성 주입은 제어 역전 기법을 의존성 관리에 적용한 메커니즘이다.
    - 제어역전에서는 한 객체가 맡은 보조 책임을 새로운 객체에게 전적으로 떠넘긴다. 새로운 객체는 넘겨받은 책임만 맡으므로 단일 책임원칙을 지키게 된다. 의존성 관리 맥락에서는 객체는 의존성 자체를 인스턴스로 만드는 책임은 지지 않는다.
    - 대신 이런 책임을 다른 전담 메커니즘에 넘겨야만 한다. 그렇게 함으로써 제어를 역전
    - 진정한 의존성 주입은 한걸음 더 나가는데 클래스가 의존성을 해결하려 시도하지 않는다. 클래스는 수동적이기에 의존성을 주입하는 방법으로 설정자 메서드나 생성자 인수를 제공한다.
    - 스프링 프레임워크는 자바 di 컨테이너를 제공한다. 의존성은 xml파일에 정의한다.
* 확장
  + 처음부터 올바르게 시스템을 만들 수 있다는 믿음은 미신이다. 대신에 우리는 오늘 주어진 사용자 스토리에 맞춰 시스템을 구현해야 한다. 내일은 새로운 스토리에 맞춰 시스템을 조정하고 확장하면 된다.
  + 이것이 반복적이고 점진적인 애자일 방식의 핵심이다. 테스트 주도 개발, 리펙터링, 꺠끗한 코드는 코드 수준에서 시스템을 조정하고 확장하기 쉽게 만든다.
  + 소프트웨어 시스템은 물리적인 시스템과 다르므로 관심사를 적절히 분리해 관리한다면 소프트웨어 아키텍쳐는 점진적으로 발전할 수 있다. 소프트웨어 시스템은 수명이 짧다는 본질로 인해 아키텍쳐의 점진적인 발전이 가능하다.
  + 횡단 관심사
    - 원하는 트랜잭션 보안 일부 영속적인 동작은 소스코드가 아니라 배치 기술에서 정의
    - 원론적으로 모듈화되고 캡슐화된 방식으로 영속성 방식을 구상할 수 있다. 현실적으로 영속성 방식을 구현한 코드가 온갖 객체로 흩어진다.
* 자바 프록시
  + - 자바 프록시는 단순한 상화에 적합하다. 개별 객체나 클래스에서 메서드 호출을 감싸는 경우가 좋은 예. 하지만 동적 프록시는 인터페이스만 지원
* 순수 자바 AOP 프레임워크
  + - 다행스럽게도 대부분의 프록시 코드는 판박이라 도구로 자동화 할 수 있다. 순수 자바 관점으로 구현하는 AOP .JBOSS AOP등과 같은 여러 자바 프레임워크는 내부적으로 프록시를 사용한다.
    - 스프링은 비즈니스 논리를 POJO로 구현하고 POJO는 순수하게 도메인에 초점을 맞춘다.
    - 프로그래머는 설정 파일이나 API를 사용해 필수적인 애플리케이션 기반 구조를 구현한다. 여기에는 영속성 트랜잭션 스프링이나 JBOSS 라이브러리의 관점을 명시한다.
* AspectJ 관점
  + 관심사를 관점으로 분리하는 가장 강력한 도구는 AspectJ언어이다(닉값을한다.)
  + 언어차원에서 관점을 모듈화 구성으로 지원하는 자바 언어 확장
  + 스프링 AOP와 JBOSS AOP가 제공하는 순수 자바 방식은 관점이 필요한 상황중 80~90에 충분
* 테스트 주도 시스템 아키텍쳐 구축
  + 관심사를 분리하는 방식은 그 위력이 막강하다. 애플리케이션 도메인 논리를 POJO로 작성할 수 있다면 코드 수준에서 아키텍쳐 관심사를 분리할 수 있다면 진정한 테스트 주도 아키텍쳐를 복잡한 아키텍쳐로 키워갈 수있다.
* 의사 결정을 최적화하라
  + 모듈을 나누고 관심사를 분리하면 지엽적인 관리와 결정이 가능
  + 가장 적합한 사람에게 책임을 맡기면 가장 좋다.
* 명백한 가치가 있을 때 표준을 현명하게 사용하라
* 시스템은 도메인 특화 언어가 필요하다
  + 좋은 DSL(도메인 특화 언어)는 도메인 개념과 그 개념을 구현한 코드 사이에 존재하는 의사소통간격을 줄여준다. 에자일 기법이 팀과 프로젝트 이해 관계자 사이에 의사소통간격을 줄여주듯
* 결론
  + 시스템 역시 꺠끗해야 한다..
  + 깨끗하지 못한 아키텍쳐는 도메인 논리를 흐리며 기민성을 떨어뜨린다. 도메인 논리가 흐려지면 제품 품질이 떨어진다.
  + 시스템을 설계하든 개별 모듈을 설계하든 실제로 돌아가는 가장 단순한 수단을 사용해야 한다는 사실을 명심하자.