* SOLID원칙, OOP(객체지향 프로그래밍) 설계 원칙

1. 단일 책임 원칙(SRP : Single Responsibility Principle)

객체는 단 하나의 책임만 가져야 한다.

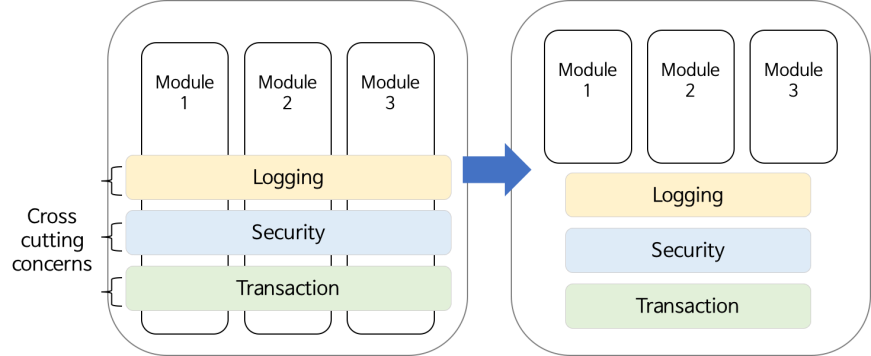
* 1. 책임이란
     1. 해야하는 것, 할 수 있는 것, 해야하는 것을 잘 알 수 있는 것
     2. 객체에 책임을 할당할 때는 어떤 객체보다는 작업을 잘 할 수 있는 객체에 책임을 할당해야 한다.
  2. 설계 원칙을 학습하는 이유
     1. 예측하지 못한 변경사항이 발생하더라도 유연하고 확장성이 있도록 시스템 구조를 설계하기 위해서
  3. 좋은 설계란
     1. 기본적으로 시스템에 새로운 요구사항이나 변경이 있을 때 가능한 한 영향 받는 부분을 줄이도줄 하는 것
     2. 즉 객체가 변해야 하는 이유는 단 1개

책임분리

* + 한 클래스에 단 하나의 책임만 수행하도록 해 변경 사유가 될 수 있는 것을 하나로 만들어야 한다.
    - 책임을 많이 질수록 클래스 내부에서 서로 다른 역할을 수행하는 코드끼리 강하게 결합될 가능성이 높아진다.

산탄총 수술

* + 하나의 책임이 여러 개의 클래스들로 분산되어 있는 경우에도, 단일 책임 우너칙에 임각해 설계를 변경해야 하는 경우도 있다.
    - 예를들어 로깅, 보안 트랜잭션과 같은 횡단 괸심으로 분류 할 수 있는 기능
    - 횡단 관심에 속하는 기능은 대부분 시스템 핵심 기능(하나의 책임) 안에 포함되는 부가 기능(여러 클래스로 분리)이다.
  + 부가 기능을 별개의 클래스로 분리해 책임을 담당하게 한다.
    - 즉 여려 곳에 흩어진 공통 책임을 한 곳에 모으면서도 응집도를 높인다.
    - 그러나 이런 독립 클래스를 구현하더라도 구현된 기능들을 호출하고 사용하는 코드가 해당 기능을 사용하는 코드 어딘가에 포함될 수 밖에 없다.



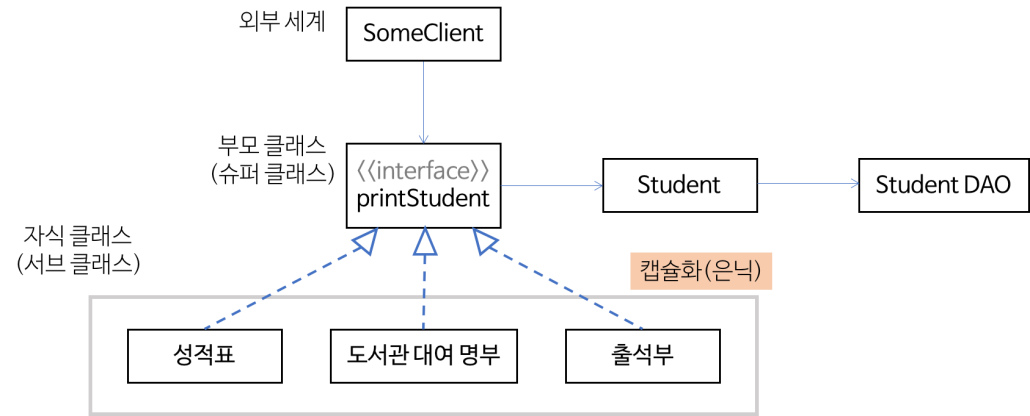
관심지향 프로그래밍(AOP)과 횡단 괸심 문제

* + 횡단 관심 문제(Cross Cutting Concern)를 해결하는 방법
    - 관심 지향 프로그래밍 기법(AOP., Aspect Oriented Programming)
  + AOP란
    - 횡단 관심을 수행하는 코드는 aspect라는 특별한 객체를 모듈화 하고 weaving이라는 작업을 통해 모듈화한 코드를 핵심 기능에 끼워넣을 수 있다.
    - 이를 통해 기존의 코드를 전혀 변경하지 않고도 시스템 핵심 기능에서 필요한 부가 기능을 효과적으로 이용할 수 있다.
    - 만약 횡단 관심에 변경이 많다면 aspect만 수정하면 된다.
  + AOP관련 용어
    - JoinPoint : 애플리케이션 실행중의 특정한 지점
    - Advice : 특정 JoinPoint에 실행하는 코드
    - PointCut : 여러 JoinPoint의 집합체로, 언제 Advice를 실행할 지 정의할 때 사용
    - Aspect : 애플리케이션이 가져야 할 로직과 그것을 실행해야 하는 지점을 정의한 것
    - Weaving : 애플리케이션 코드의 해당 지점에 aspect를 실제로 주입하는 과정

1. 개방 폐쇄 원칙(OCP, Open Closed Principle)

기존의 코드를 변경하지 않으면서 기능을 추가할 수 있도록 설계가 되어야 한다.

* + 설계의 기본(의존 역전 원칙(DIP)과 밀접)이라고 할 수 있다.
  + OCP를 만족하는 설계 예시



* + - 인터페이스에서 구체적인 출력 매체를 캡슐화해 처리하도록 해야한다.
  + OCP에 위반하지 않은 설계를 할 떄 가장 중요한 것은 무엇이 변하는 것인지 무엇이 변하지 않는 것인지를 구분해야 한다는 점이다.
    - 변해야 하는 것은 쉽게 변할 수 있게 하고, 변하지 않아야 할 것은 변하는 것에 영향을 받지 않게 해야 한다.
  + 클래스를 변경하지 않고도 대상 클래스의 환경을 변경할 수 있는 설계가 되어야 한다.
    - 특히 단위 테스트를 수행할 떄 매우 중요하다.

1. 리스코프 치환 원칙(LSP, Liskov Substitution Principle)

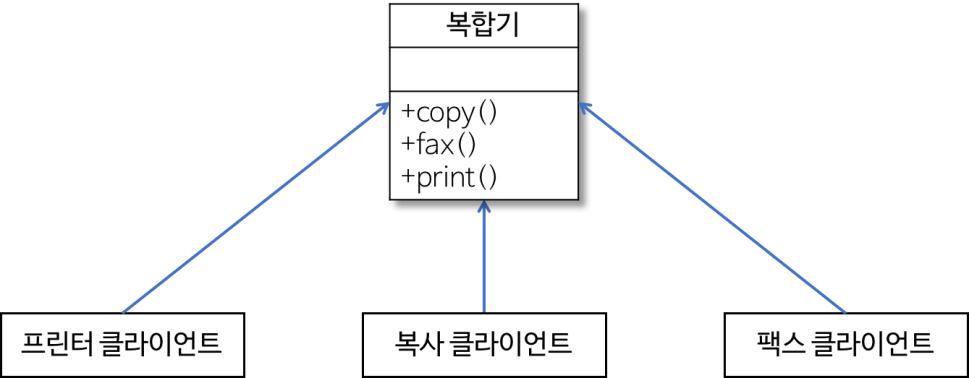
일반화 관계에 대한 이야기이며, 자식클래스는 최소한 자신의 부모 클래스에서 가능한 행위는 수행할 수 있어야 한다..

* LSP는 부모클래스와 자식 클래스 사이의 행위가 일관성이 있어야 한다는 의미
  + 즉 LSP를 만족하면 프로그램에서 부모 클래스의 인스턴스 대신에 자식 클래스의 인스턴스로 대체해도 프로그램의 의미는 변화하지 않는다.
  + 일반화 관계
    - Is a kind of 의 관계
    - 예를들어 원숭이 is kind of 포유류
    - 포유류는 부모클래스 원숭이는 자식클래스라고 보면 된다.
  + LSP를 만족시키는 간단한 방법은 재정의하지 않는 것
    - 즉 부모 클래스에서 상속받은 메소드들이 자식 클래스에 오버라이드, 즉 재정의되지 않도록 하면 된다
    - OOP의 특징에서 언급했던 피터 코드의 상속규칙의 자식 클래스가 부모 클래스의 책임을 무시하거나 재정의 하지 않고 확장만 수행한다. 라는 규칙을 지키는 것은 LSP를 만족시키는 하나의 방법에 해당한다.

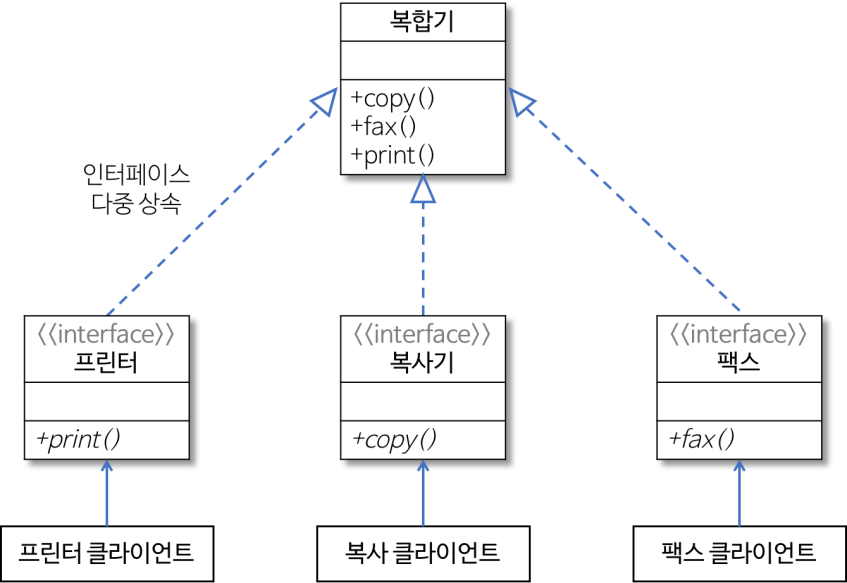
1. 인터페이스 분리 원칙(ISP, Interface Segregation Principle)

인터페이스를 클라이언트에 특화되도록 분리시키라는 설계 원칙이다.

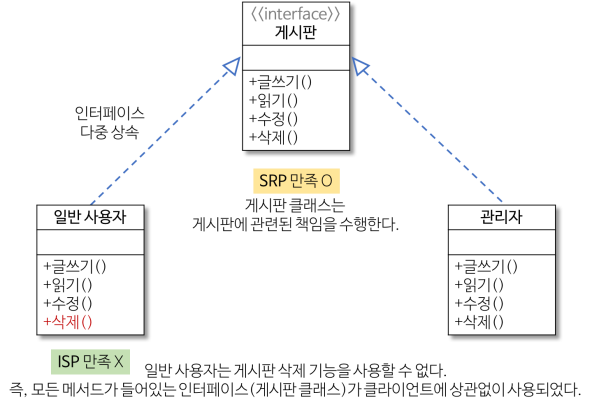
* + 클라이언트 자신이 이용하지 않는 기능에는 영향을 받지 않아야 한다.
  + 단일 책임 원칙(SRP)과 밀접하다
  + 예시
    - 복합기의 클래스 다이어그램



* + - * 복합기 기능을 제공하는 클래스는 매우 비대해질 가능성이 크다
      * 하지만 복합기의 모든 기능을 클라이언트가 동시에 사용하는 경우는 거의 없다
      * 팩스 기능의 변경으로 인해 프린터 기능만 이용하는 클라이언트가 영향을 받을 수 있다.
      * 이렇게 클라이언트와 무관하게 발생한 변화로 클라이언트 자신이 영향을 받지 않으려면 범용의 인터페이스보다는 클라이언트에 특화된 인터페이스를 사용해야 한다.
    - ISP에 만족하는 복합기 클래스 다이어그램



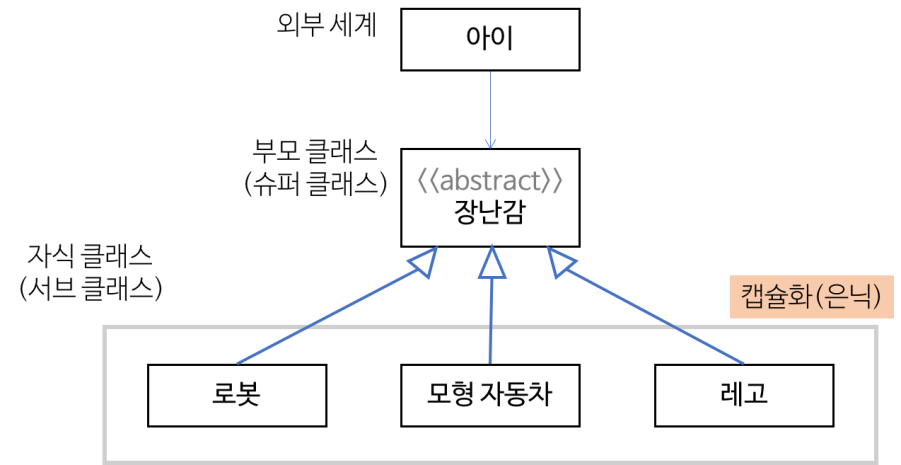
* + - * 복합기를 사용하는 객체들마다 자신이 관심을 갖는 메서드들만 있는 인터페이스를 제공받도록 설계했다
      * 인터페이스가 일종의 방화벽 역할을 수행
      * 클라이언트는 자신이 사용하지 않는 메소드에 생긴 변화로 인한 영향을 받지 않게 된다.
  + SRP와 ISP의 관계
    - 어떤 클래스가 여러 책임을 수행하게 되면 방대한 메서드를 가진 비대한 클래스가 될 가능성이 커짐
      * 이렇게 비대한 클래스를 단일 책임을 갖는 여러 클래스로 분할하면 SRP를 만족한다.
      * 또한 각자의 인터페이스를 제공한다면 ISP를 만족할 수 없다
    - 그렇다면 ISP 는 SRP를 만족하면 성립되는가
      * 반드시 그렇다고는 볼 수 없다
      * 예시



1. 의존 역전 원칙(DIP, Dependency Inversion Principle)

의존 관계를 맺을 떄 변화하기 쉬운 것 또는 자주 변화하는 것보다 변화하기 어려운 것, 거의 변화가 없는 것에 의존하려는 것

* + 객체 사이에 서로 도움을 주고 받으면 의존 관계가 발생하는데, DIP는 이떄의 가이드라인에 해당한다.
    - 클래스간 의존관계란
      * 한 클래스가 어떤 기능을 수행하려고 할 떄 다른 클래스의 서비스가 필요한 경우
  + OCP가 되려면 기본적으로 DIP가 만족되어야 한다.
  + 그렇다면 변하기 쉬운 것과 변하기 어려운 것은 어떻게 구분하는가
    - 변화하기 어려윤 것, 거의 변화가 없는 것 : “정책”, “전략”
    - 변하기 쉬운 것 : “구체적인 방식”,”사물”
  + DIP를 만족하는 설계 예시



* + - 객체지향 관점에서 변하기 어려운 추상적인 것들을 표현하는 수단
      * 추상 클래스
      * 인터페이스
  + DIP를 만족시키는 방법
    - 어떤 클래스가 도움을 받을 때 구체적인 클래스보다는 인터페이스나 추상 클래스와 의존관계를 맺도록 설계해야 한다.
  + DIP를 만족하는 설계는 변화에 유연한 시스템이 된다
    - 의존성 주입(DI , Dependency Injection) 이라는 기술로 변화를 쉽게 수용할 수 있는 코드를 작성할 수 있다.
    - 의존성 주입이란?
      * 클래스 외부에서 의존되는 것을 대상 객체의 인스턴스 변수에 주입하는 기술
    - Hollywood prinple(할리우드 원칙)
      * Don’t call me, we’ll call you
      * We : 프레임워크
      * You : 구현 모듈, 필요 객체
      * 즉 아이가 마음이 바뀌어 다른 어떤 장난감을 가지고 놀더라도 기존의 코드에 전혀 영향을 받지 않고도 장난감을 바꿀 수 있다는 뜻