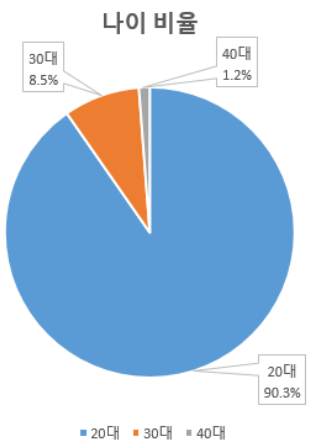
[Project News] 코드 설계를 위한 지식, 디자인 패턴

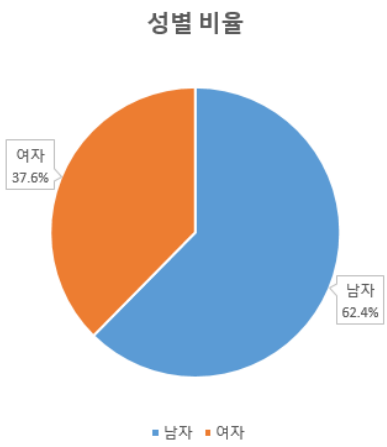
- 진행: 황수재 컨설턴트님
- 날짜: 2020.10.26 (월요일)
- 목차
  - 서울 4반 1팀 프로젝트 설문조사
  - 라이브 방송 일정
  - OOP 원칙: SOLID
  - 디자인 패턴

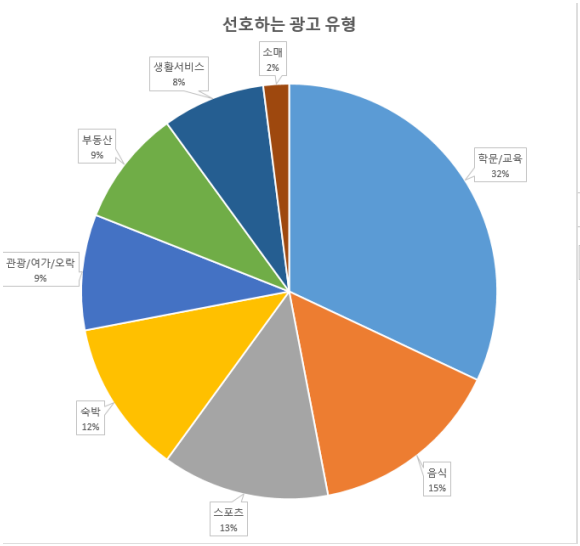
서울 4반 1팀 프로젝트 설문조사

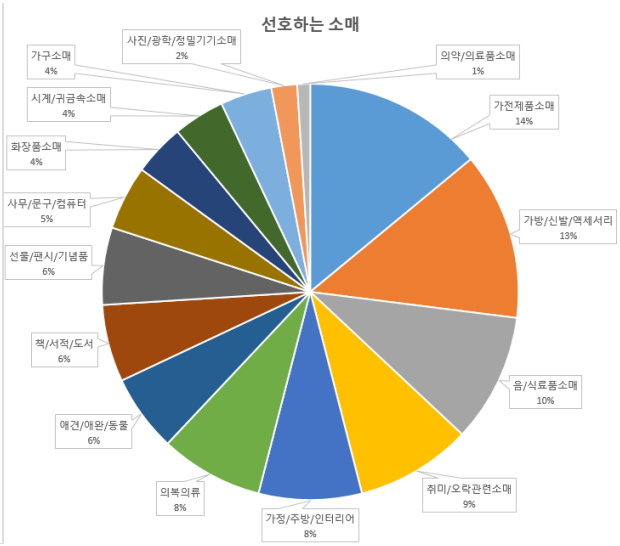
> 프로젝트: '돌리 Go!'

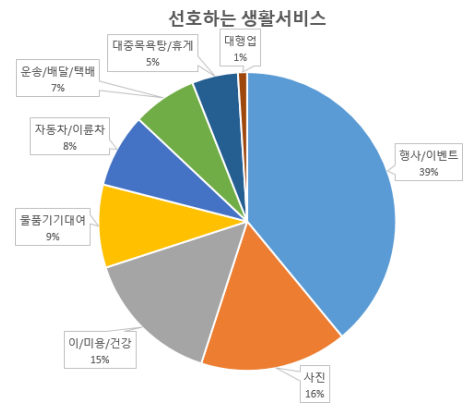
- 서울 4반 1팀 '또 나만 진심이지' 팀에서 요청한 설문조사 입니다.
- '모바일 전단지'를 주제로 프로젝트를 진행하고 있습니다.
- 디지털 모바일 전단지 서비스 '돌리 Go!'를 기획하는 중 사용자 관점에서의 광고 선호도를 조사하여 프로젝트에 반영하고자 SSAFY 3기, 4기 교육생을 대상으로 설문조사를 실시하였습니다.

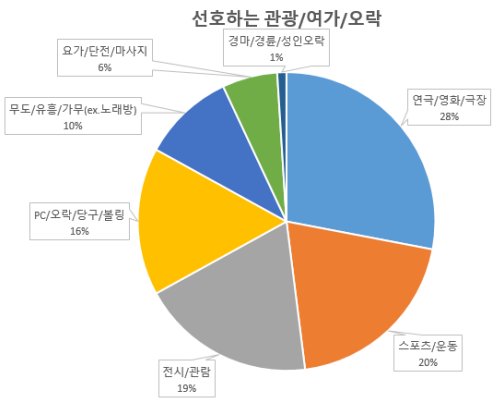


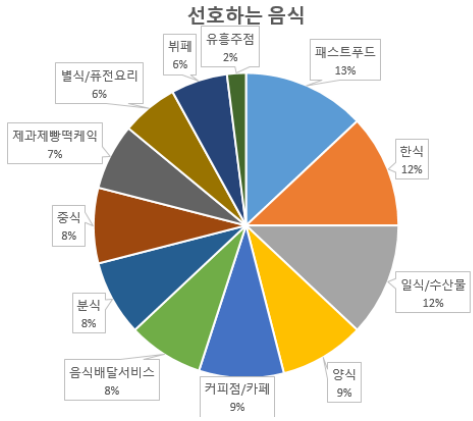


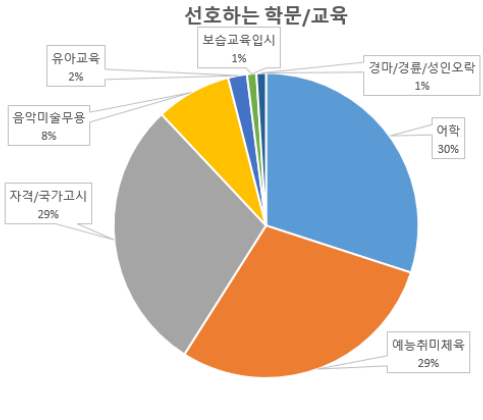

선호하는 광고 유형
소매 2%
생활서비스 8%
부동산 9%
관광/여가/오락 9%
숙박 12%
스포츠 13%
음식 15%
학문/교육 32%


선호하는 소매
사진/광학/정밀기기소매 2%
가구소매 4%
시계/귀금속소매 4%
화장품소매 4%
사무/문구/컴퓨터 5%
선물/팬시/기념품 6%
책/서적/도서 6%
애견/애완/동물 6%
의복의류 8%
가정/주방/인터리어 8%
취미/오락관련소매 9%
음/식료품소매 10%
가방/신발/액세서리 13%
가전제품소매 14%
의악/의료품소매 1%

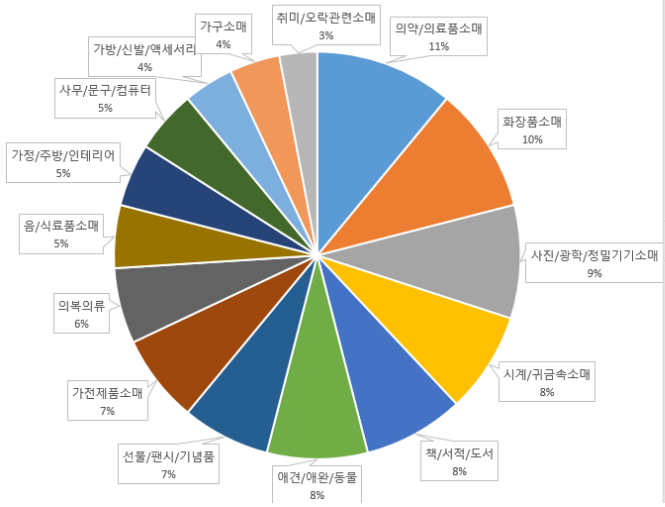

선호하는 생활서비스
대행업 1%
대중목욕탕/휴게 5%
운송/배달/택배 7%
자동차/이륜차 8%
물품기기대여 9%
이/미용/건강 15%
사진 16%
행사/이벤트 39%


선호하는 관광/여가/오락
요가/단전/마사지 6%
경마/경륜/성인오락 1%
무도/유흥/가무(ex.노래방) 10%
PC/오락/당구/볼링 16%
전시/관람 19%
스포츠/운동 20%
연극/영화/극장 28%

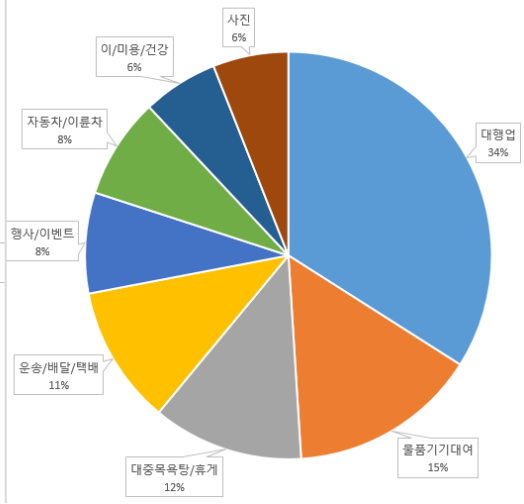

선호하는 음식
뷔페 6%
유흥주점 2%
패스트푸드 13%
별식/퓨전요리 6%
제과제빵떡케익 7%
한식 12%
중식 8%
일식/수산물 12%
분식 8%
음식배달서비스 8%
커피점/카페 9%
양식 9%


선호하는 학문/교육
보습교육입시 1%
유아교육 2%
경마/경륜/성인오락 1%
음악미술무용 8%
어학 30%
자격/국가고시 29%
예능취미체육 29%

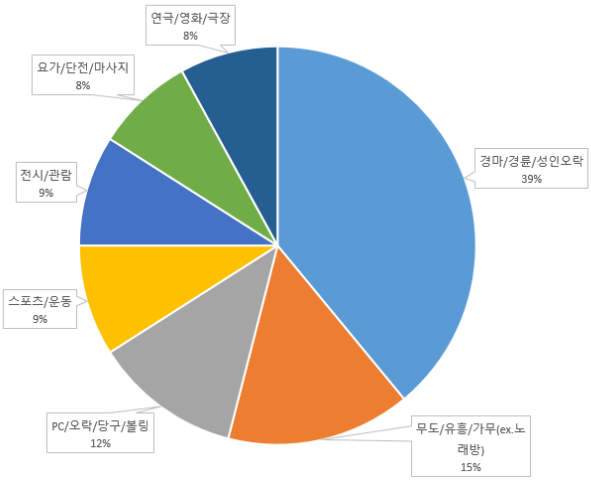
싫어하는 소매



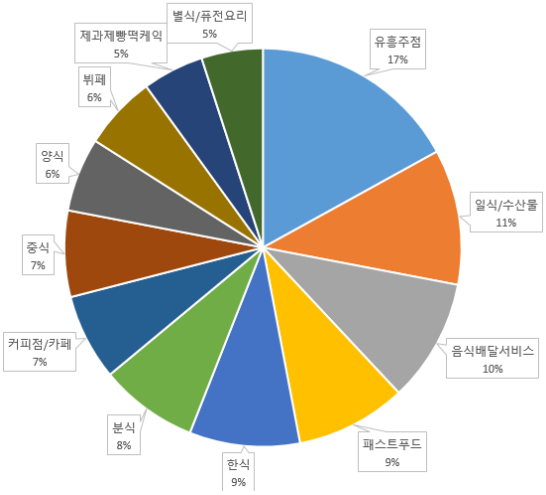
싫어하는 생활서비스



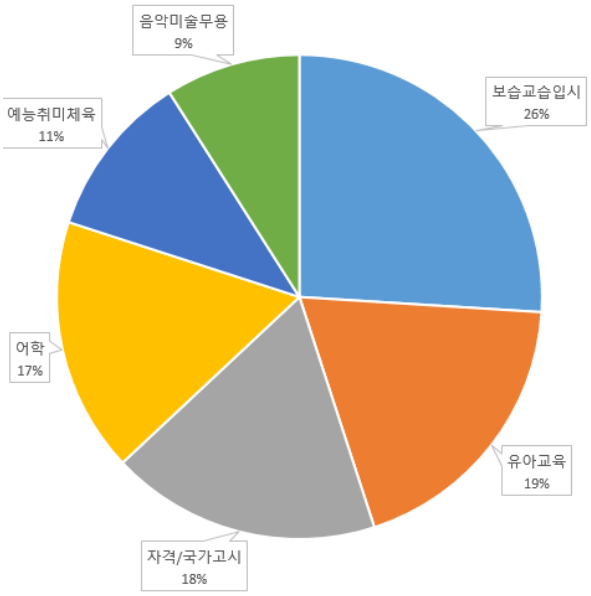
싫어하는 관광/여가/오락



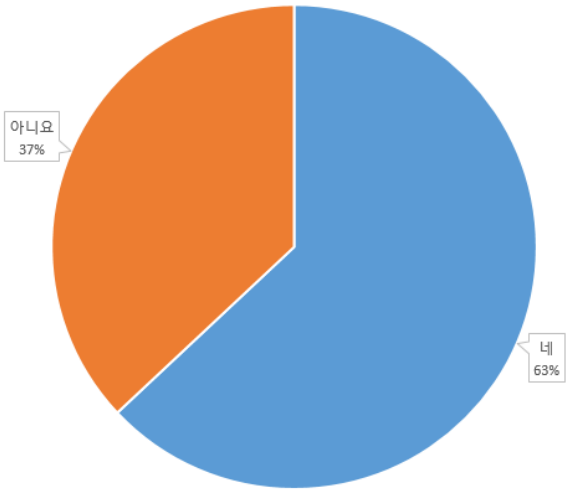
싫어하는 음식

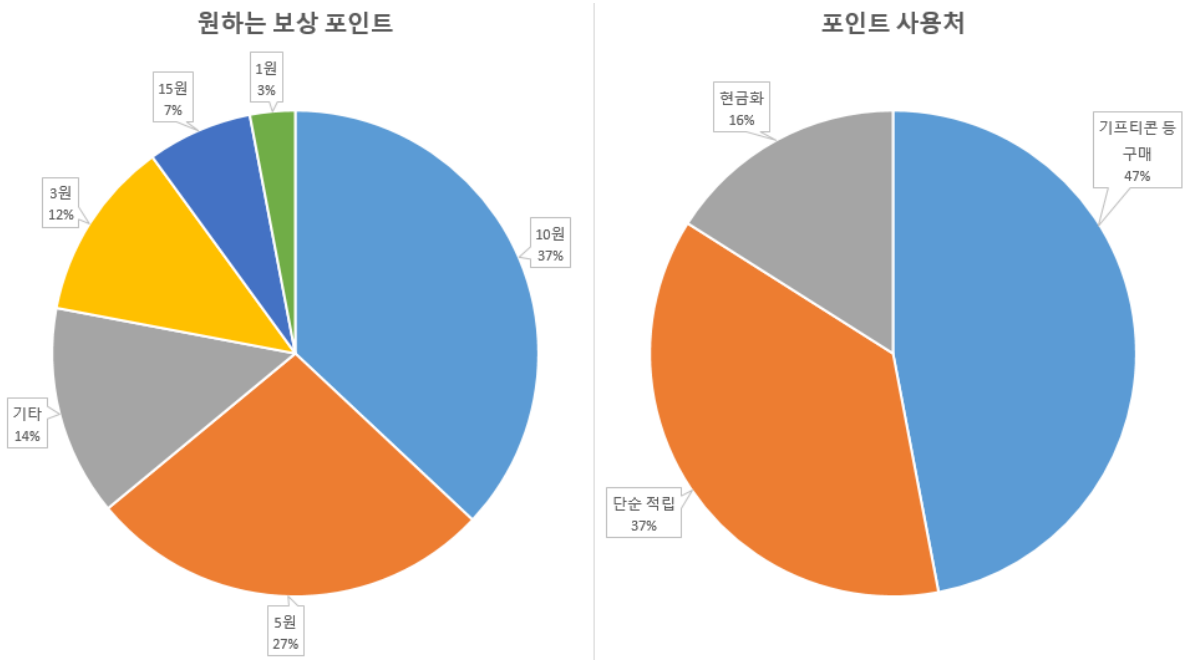


싫어하는 학문/교육



리워드 어플 사용여부



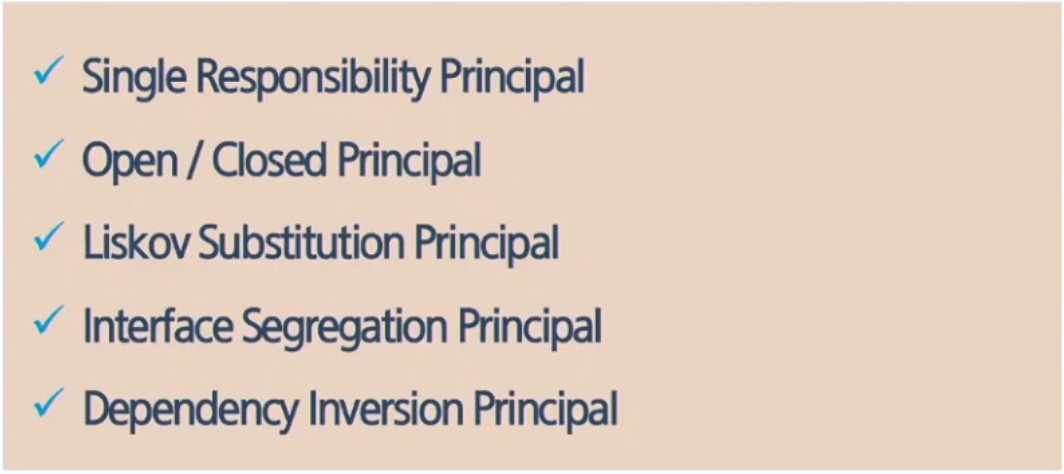


라이브 방송 일정

주차	일자	요일	시간	구분	주요내용
3주차	10월 26일	월	9시	Project News	디자인 패턴
3주차	10월 28일	수	9시	IT Essential	클라우드의 이해
3주차	10월 30일	금	9시	Project Review	3기 특화 프로젝트 리뷰

OOP 원칙: SOLID

> 객체지향 5대 원칙



- **S**ingle Responsibility Principle (단일 책임 원칙)
  - 객체지향 프로그래밍에서 모든 클래스는 하나의 책임만 가지며, 클래스는 그 책임을 완전히 캡슐화해야 한다.

- **O**pen / Closed Principle (개방/폐쇄 원칙)
    - 소프트웨어 개체(클래스, 함수, 모듈 등)는 확장에 대해 열려 있어야 하고, 수정에 대해서는 닫혀 있어야 한다.
- **L**iskov Substitution Principle (리스코프 치환 원칙)
    - 파생 자료형은 기본 자료형과 치환할 수 있어야 한다.
      ;상속을 받은 자식클래스는 부모 클래스의 행위를 할 수 있어야 한다.
- **I**nterface Segregation Principle (인터페이스 분리 원칙)
    - 클라이언트가 자신이 이용하지 않는 메소드에 의존하지 않아야 한다.
      ;인터페이스를 구체적이고 작은 단위로 분리시켜서 클라이언트에게 꼭 필요한 메소드들만 사용할 수 있게 한다.
- **D**ependency Inversion Principle (의존관계 역전 원칙)
    - 상위 계층(정책 결정)이 하위 계층(세부 사항)에 의존하는 전통적인 의존관계를 반전(역전)시킴으로써 상위 계층이 하위 계층의 구현으로부터 독립되게 한다.
        1. 상위 모듈은 하위 모듈에 의존해서는 안된다. 상위 모듈과 하위 모듈 모두 추상화에 의존해야 한다.
        2. 추상화는 세부 사항에 의존해서는 안된다. 세부사항이 추상화에 의존해야 한다.

디자인 패턴

- GoF (Gang of Four) Design Pattern
  Design Patterns: Elements of Reusable Object-Oriented Software
  : 사인방(Gang of Four) Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides이 작성
- 객체지향 개발을 위한 설계 Best Practices 모음
- 3분류(생성, 구조, 행위)에서의 총 23개 패턴

디자인 패턴의 이점

- 개념화와 다이어그램으로 표현이 쉽고 해결법을 재사용 가능 (UML)
- 개발자간 커뮤니케이션 용이
- 확장성, 재사용성, 유지보수성이 좋은 소프트웨어 설계

디자인 패턴의 종류

생성	구조	행위
Abstract Factory Builder Factory method Prototype Singleton	Adaptor Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Interpreter Iterator Mediator Memento Observer State Strategy Template Method Visitor

아래에서 **bold**는 오늘 라이브 강의에서 다룬 패턴을 의미함.

생성 패턴(Creational Patterns)

- Abstract Factory(추상 팩토리 패턴)
- Builder(빌더 패턴)
- Factory method(팩토리 메소드 패턴)
- Prototype(프로토타입 패턴)
- **Singleton(싱글턴 패턴)**

구조 패턴(Structural Patterns)

- **Adaptor(어댑터 패턴)**
- Bridge(브릿지 패턴)
- Composite(컴포지트 패턴)
- Decorator(데코레이터 패턴)
- Façade(퍼사드 패턴)
- Flyweight(플라이웨이트 패턴)
- Proxy(프록시 패턴)

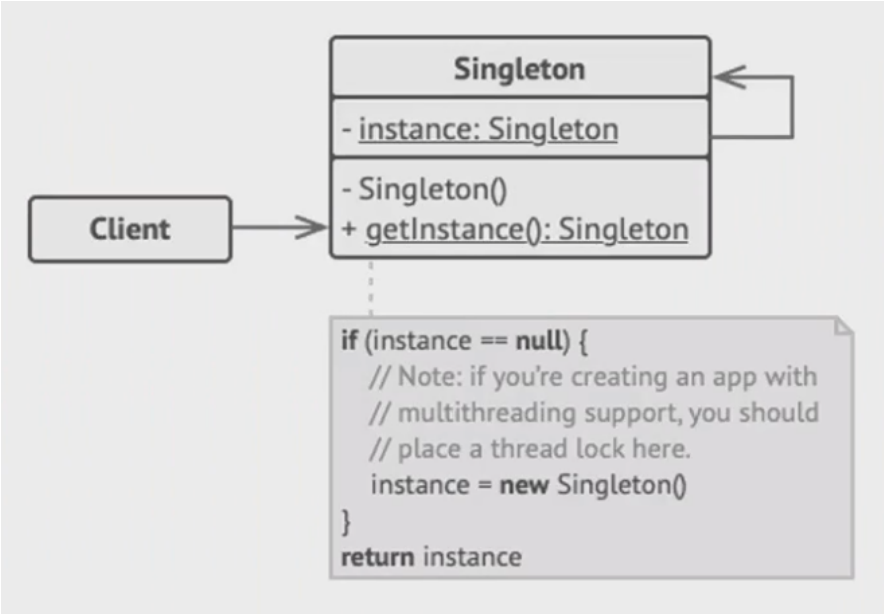
행동 패턴(Behavioral Patterns)

- Chain of Responsibility(책임 연쇄 패턴)
- Command(커맨드 패턴)
- Interpreter(해석자 패턴)
- Iterator(반복자 패턴)
- Mediator(중재자 패턴)
- Memento(메멘토 패턴)
- Observer(옵저버 패턴)
- State(상태 패턴)
- **Strategy(전략 패턴)**
- Template Method(템플릿 메소드 패턴)
- Visitor(방문자 패턴)

생성: Singleton Pattern

> 생성자가 여러 차례 호출되더라도 실제로 생성되는 객체는 하나이고, 최초 생성 이후에 호출된 생성자는 최초의 생성자가 생성한 객체를 리턴하도록 하는 유형이다.

싱글톤 패턴의 UML 표현 예시

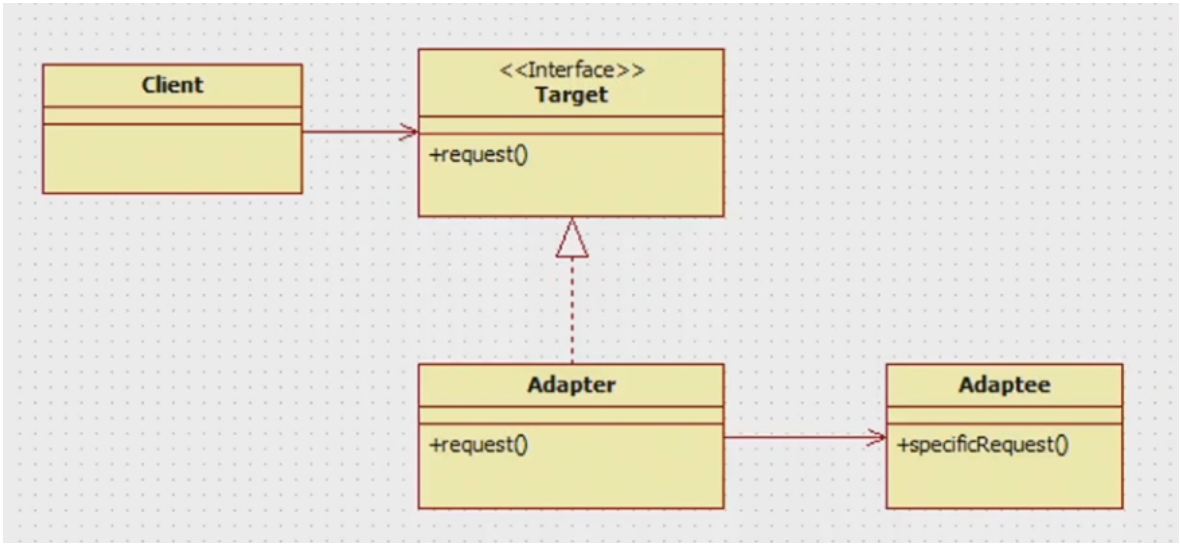



```
접근 제한자
- : private
+ : public
# : protected
-------------
Singleton Pattern : 인스턴스의 접근 및 생성을 외부에서 할 수 없도록 설계하는 방식
- instance: 인스턴스를 외부에서 접근할 수 없음
- Singleton(): 생성자도 외부에서 호출될 수 없음
+ getInstance(): 이 메소드를 이용하여 객체에 접근할 수 있음
```

DBCP(Database Connection Pool) 또는 System configuration 등 공통된 객체를 여러개 생성해서 사용하는 경우에 많이 사용됨

구조: Adapter Pattern

> 클래스의 인터페이스를 사용자가 기대하는 다른 인터페이스로 변환하는 패턴.
> 호환성이 없는 인터페이스 때문에 함께 동작할 수 없는 클래스들이 함께 동작할 수 있도록 한다.



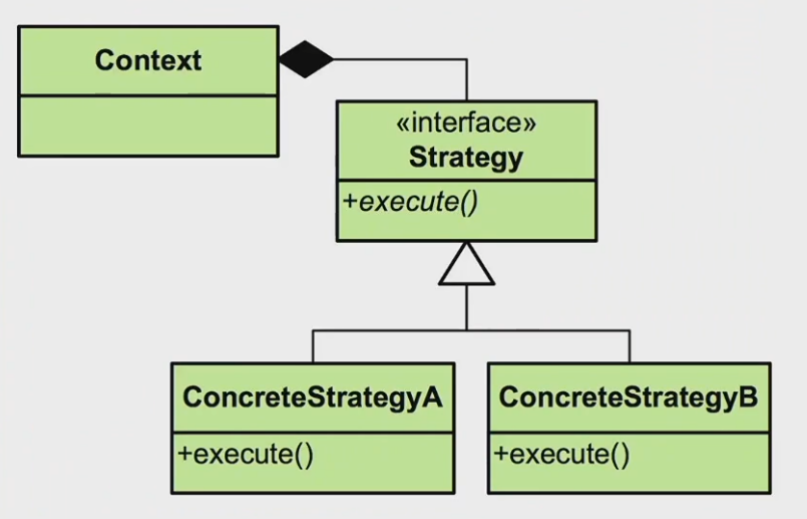
Adaptee와 Client의 인터페이스가 호환되지 않을 때, Adapter를 이용하여 재사용 할 수 있도록 한다.

- Client : Thrid party 라이브러리나 외부 시스템을 사용하는 쪽을 의미한다.
- Adaptee : Thrid party 라이브러리나 외부시스템을 의미한다.

- Target(Interface) : Adapter가 구현하는 인터페이스. 클라이언트는 Target(Interface)를 통해 Adaptee인 써드파티 라이브러리를 사용하게 된다.
- Adapter : Client와 Adaptee 중간에서 호환성이 없는 둘을 연결시켜주는 역할을 담당한다. Target(Interface)을 구현하며 클라이언트는 Target(Interface)를 통해 어댑터에 요청을 보낸다. 어댑터는 클라이언트의 요청을 Adaptee가 이해할 수 있는 방법으로 전달하고 처리는 Adaptee에서 이루어진다.

행위: Strategy Pattern

> 특정 컨텍스트에서 알고리즘을 분리하는 설계 패턴.
>
> 객체들이 할 수 있는 행위 각각에 대해 전략 클래스를 생성하고 유사한 행위들을 캡슐화하는 인터페이스를 정의하여, 객체의 행위를 동적으로 바꾸고 싶은 경우 직접 행위를 수정하지 않고 전략을 바꿔줌으로써 행위를 유연하게 확장한다.



예시 1: 컨텍스트가 *로그인* 일 때 이메일과 패스워드 로그인 또는, 카카오, 네이버, 구글 로그인으로 로그인하는 행위(각각 `ConcreteStrategyA`, `ConcreteStrategyB`, ...)

- 동일한 로그인이라는 행위에 대해서 구현 알고리즘이 변화됨

예시 2: 컨텍스트가 *취업* 이고 인터페이스가 *서류지원* 일 때 `ConcreteStrategy`는 S사, N사, K사 등과 부합하는 전략

디자인 패턴 사용시 주의사항

> *"If all you have is a hammer, everything looks like a nail"*

Maslow's Hammer 라고 알려진 이 개념은 프로그래밍에서도 전반적으로 활용할 수 있음
(;특정 도구 또는 시각에 몰입하다 보면 시야가 좁아질 수 있다는 의미로 활용됨)

자신이 Singleton pattern을 배웠으나, 이 한 가지 패턴만 알고 있다면 모든 것이 Singleton pattern에 부합하는 것처럼 보일 수 있기 때문에

- 오남용을 주의하고 폭 넓게 이해하는 것이 필요하다.
- 하나의 툴을 배웠다고 하여 모든 것에 끼워 맞추지 말고, 좀 더 적합한 툴이나 좀 더 적합한 프로그래밍 언어에 대해서 폭 넓게 이해하는 것이 필요하다.

오늘의 퀴즈

Q. 인스턴스가 오직 하나만 생성되는 것을 보장하고, 어디서든 이 인스턴스에 접근할 수 있도록 하는 디자인 패턴은?

1. Single Malt
2. Single Responsibility Principal
3. **Singleton**
4. Single Bed