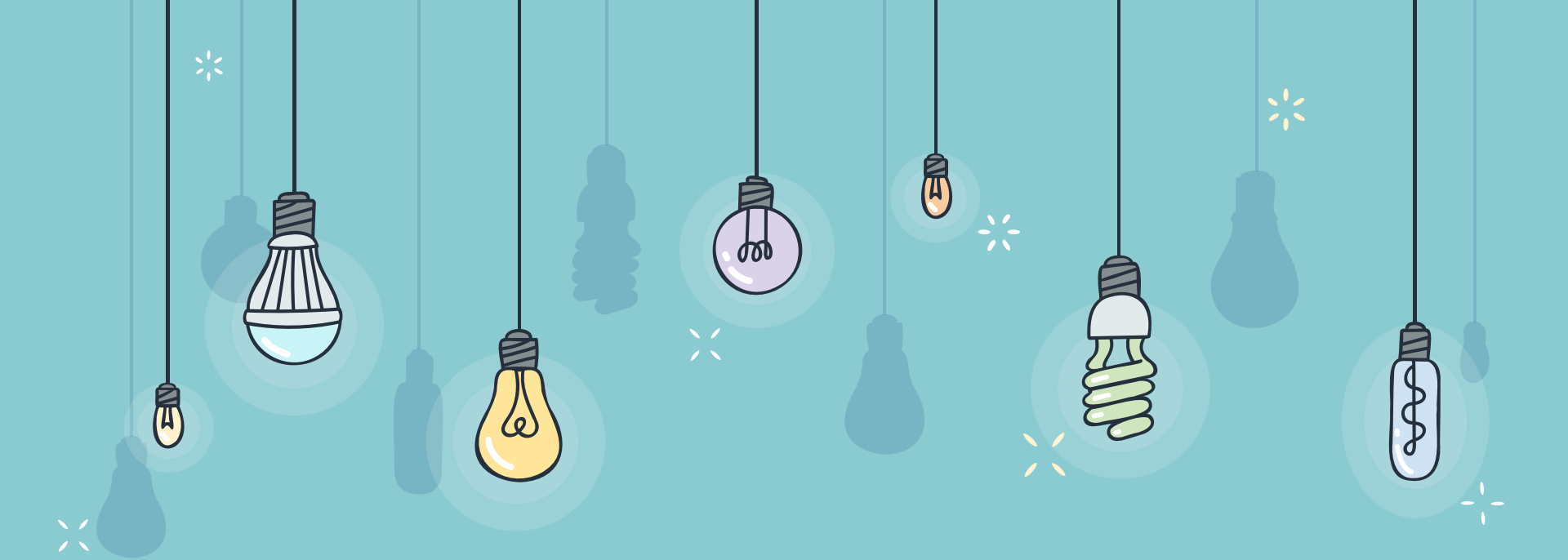
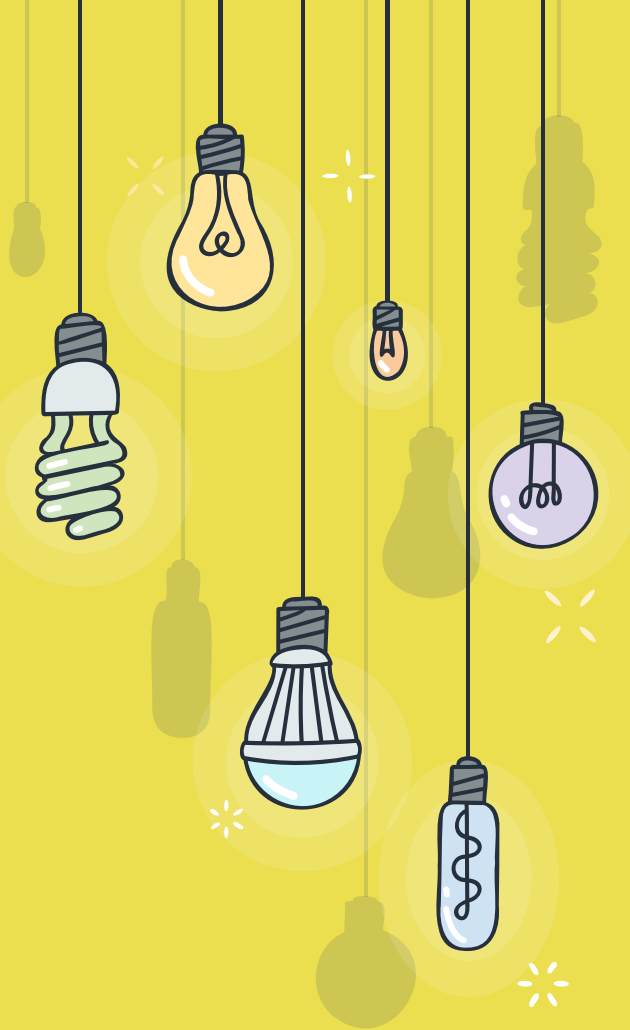


디자인 패턴

김 지원

1. 디자인 패턴 개념
2. GoF 디자인 패턴
3. Factory Pattern
4. Builder Pattern
5. Observer Pattern

1 디자인패턴의 개념



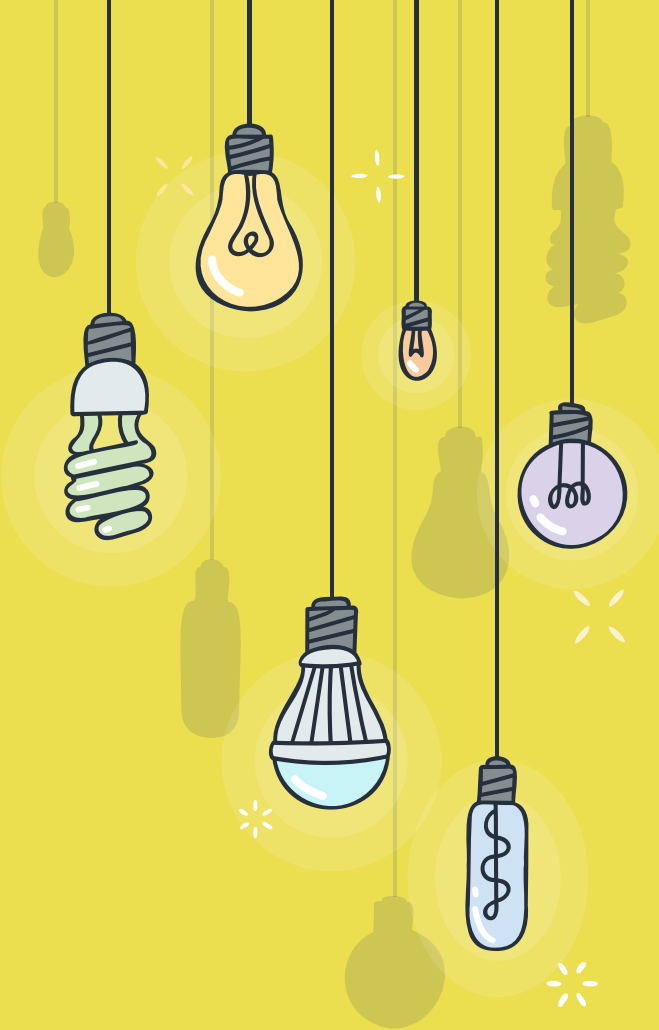
디자인 패턴의 개념

- 디자인 패턴
  - 프로그래밍에서 공통적인 문제를 쉽게 해결하기 위해 만들어진 가이드라인
    - 라이브러리처럼 실제 코드를 제공하지는 않지만, 특정 유형의 문제를 해결하는 방식을 제공. 즉, 공식
  - 크리스토퍼 알렉산더
    - 바퀴를 다시 발명하지 마라 (Dont reinvent the wheel)
    - 패턴은 비슷하거나 동일한 양식 또는 유형들이 반복되어 나타난다는 의미
    - 문제와 해결책도 동일한 유형이나 양식을 통해 쉽게 찾을 수 있다.

왜 필요한가

- 디자인 패턴이 필요한 이유
  - 재사용성
    - 개발자는 문제가 발생할 때마다 코드를 다시 구현하는 대신 기존 솔루션을 활용할 수 있다.
  - 유지보수성
    - 문제를 해결하는 표준 방법을 제공하여 코드를 보다 유지보수하기 쉽게 해준다
    - 개발자는 해당 패턴에 익숙하므로(학습을 하였다면) 코드를 이해하고 수정하기가 더 쉽다.
  - 모범 사례
    - 일반적인 문제에 대한 모범 사례와 검증된 해결책을 기반으로 만들어 졌다.
    - 문제에 대한 효과적이고 효율적인 솔루션을 사용할 수 있다.

2 GOF Design Pattern



GoF

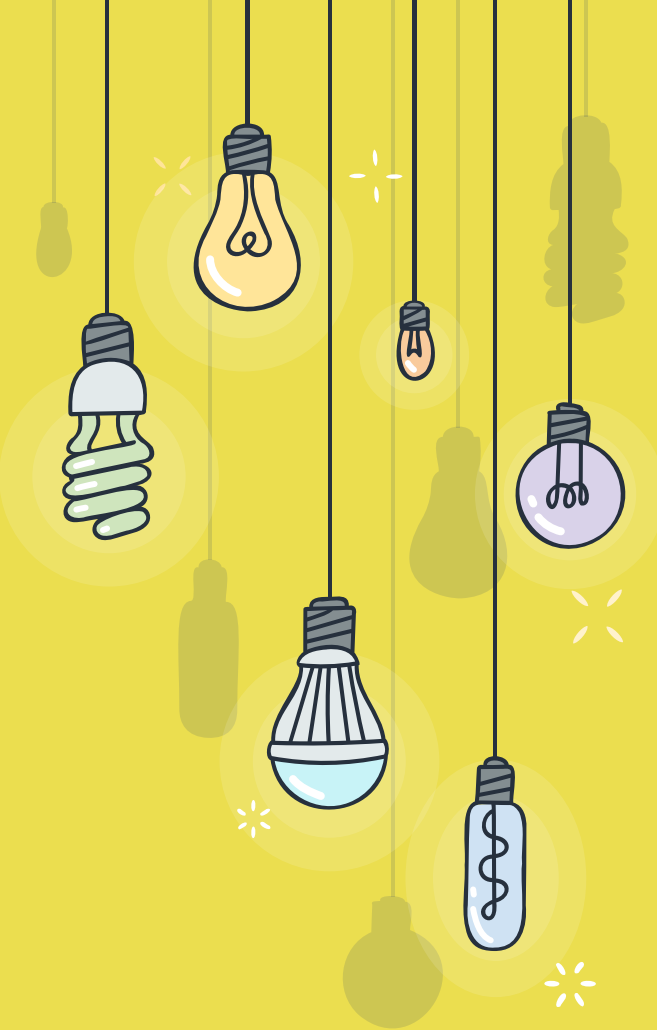
- GoF
  - 감마 외 3인(Gang of Four)이 쓴 디자인 패턴 책(23개 패턴)

  - 디자인 패턴의 분류기준
  - 목적에 따른 분류
    - 각각의 패턴이 어떤 일을 하기 위한 것인지에 관한 것
    - 생성, 구조, 행동 3가지로 나눔
  - 범위에 따라 분류
    - 클래스에 적용하는지, 객체에 적용하는 지 구분하는 것
      - 클래스 패턴 : 클래스와 서브클래스 간의 관련성을 다룬다. 주로 상속을 통해 관련. 컴파일 타임에 정적으로 결정
      - 객체 패턴 : 객체 간의 관련성. 런타임에 변경될 수 있는 동적인 성격

GOF

목적 / 범위	생성(Creation)	구조(Structure)	행위(Behavior)
클래스	Factory Method	Adapter(class)	Interpreter Template Method
객체	Abstract Factory Builder Prototype Singleton	Adapter(Object) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

3 Factory Pattern



Factory pattern

- Factory Pattern
  - 어떤 클래스의 인스턴스를 만들지는 서브 클래스에서 결정하게 만드는 패턴
  - 생성패턴의 중요한 이슈
    - 시스템이 어떤 Concrete Class를 사용하는지에 대한 정보를 캡슐화 한다.
    - 생성 패턴은 이들 클래스의 인스턴스들이 어떻게 만들고 어떻게 결합하는지에 대한 부분을 완전히 가려준다

Factory ex

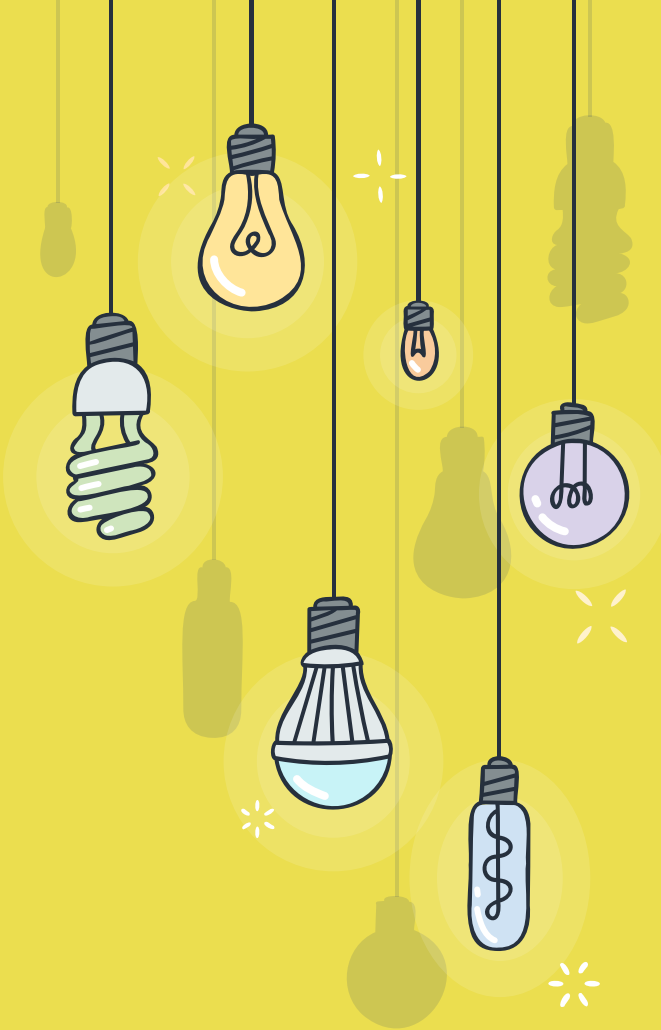
```java
public class ProductRun {
    public static void main(String[] args) {
        Product p1 = new Product();
        p1.setName("Tv");
        System.out.print(p1);

        Product p2 = new Product();
        p2.setName("Computer");
        System.out.print(p2);
    }
}
```

```java
public class Product{
    private String name;
    private int price;
    public void setName(String name) {
        this.name = name;
        if(name.equalsIgnoreCase("tv")) {
            price = 300;
        } else if(name.equalsIgnoreCase("computer")) {
            price = 200;
        }
    }

    @Override
    public String toString() {
        return "Product [name=" + name + ", price=" + price + "]";
    }
}
```

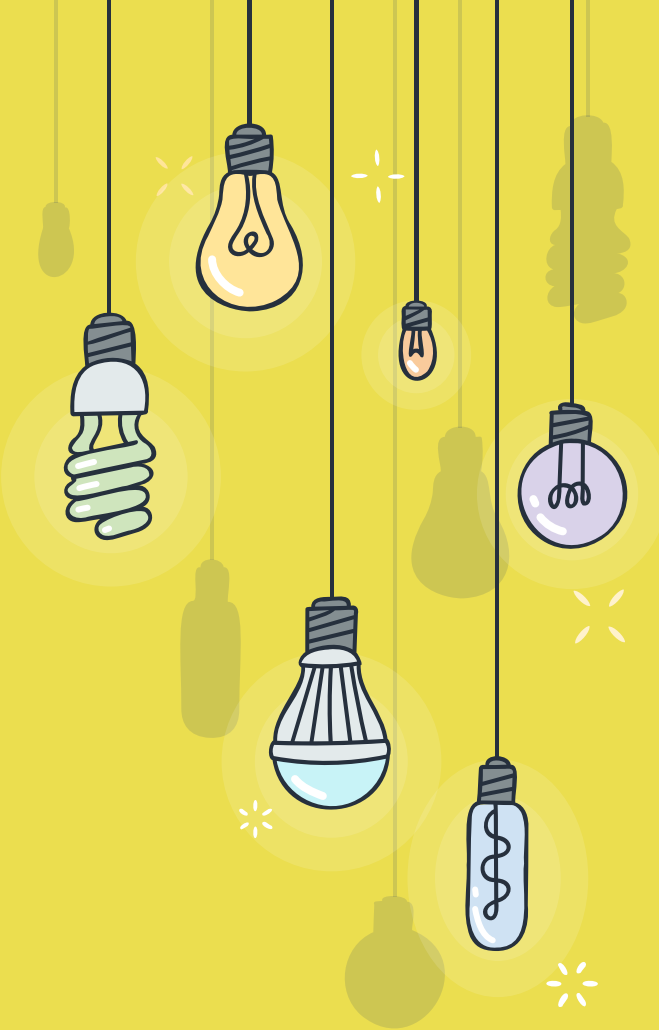
4 Builder Pattern



Builder Pattern

- Builder Pattern
  - 생성시 지정해야 할 인자가 많을 때 사용하는 패턴
    - 일반적으로 많이 사용함
  - 객체 생성시 여러 단계를 순차적으로 거칠 때 그 단계의 순서를 결정해 두고 각 단계를 다양하게 구현할 수 있도록 하는 패턴

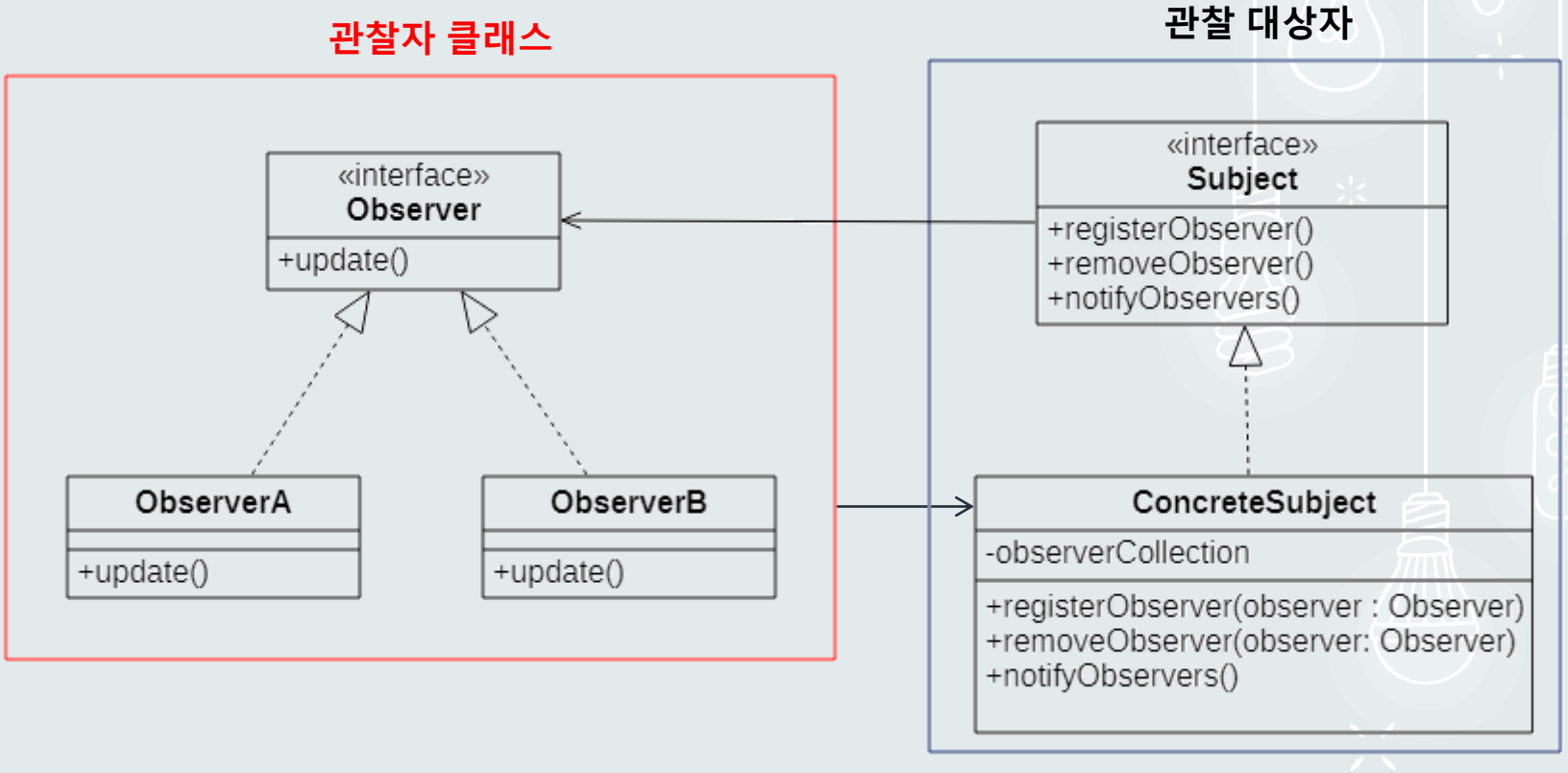
5 Observer Pattern



Observer Pattern

- Observer Pattern
  - 감시자 패턴
    - 어떤 객체의 상태가 변할 때 그와 연관된 객체들에게 알림을 보내는 디자인 패턴
  - 객체 생성시 여러 단계를 순차적으로 거칠 때 그 단계의 순서를 결정해 두고 각 단계를 다양하게 구현할 수 있도록 하는 패턴

Observer Pattern



Thanks!

- Any questions?