5주차 발표_민서연

디자인 패턴

01 디자인 패턴

디자인 패턴

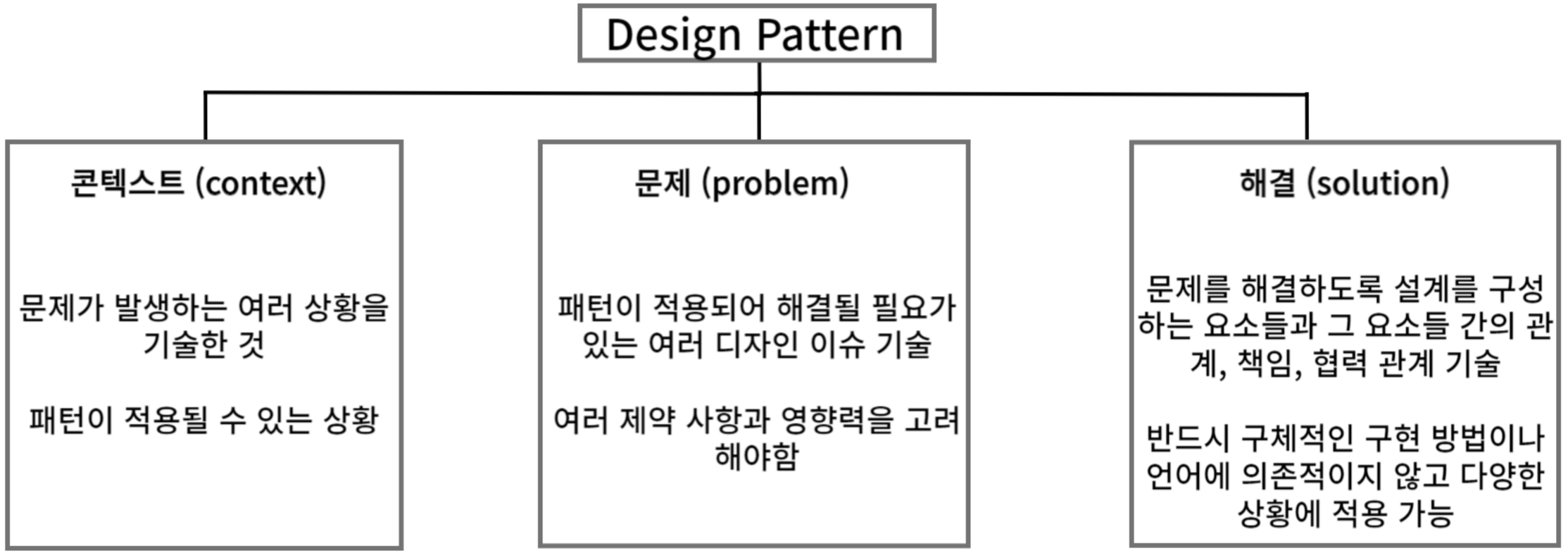
특정 문맥에서 공통적으로 발생하는 문제에 대해 재사용 가능한 해결책

이때 패턴이란?

서로 다른 소프트웨어 모듈이나 기능을 가진 다양한 응용 소프트웨어를 개발할 때 공통되는 설계문제나 해결책이 존재하는데 이러한 유사점을 뜻함

01 디자인 패턴

구조


Design Pattern
콘텍스트 (context)
문제가 발생하는 여러 상황을 기술한 것
패턴이 적용될 수 있는 상황
문제 (problem)
패턴이 적용되어 해결될 필요가 있는 여러 디자인 이슈 기술
여러 제약 사항과 영향력을 고려해야함
해결 (solution)
문제를 해결하도록 설계를 구성하는 요소들과 그 요소들 간의 관계, 책임, 협력 관계 기술
반드시 구체적인 구현 방법이나 언어에 의존적이지 않고 다양한 상황에 적용 가능

01 디자인 패턴

종류

생성 패턴 (Creational)

추상 팩토리

팩토리 메서드

싱글턴

구조 패턴 (Structural)

컴퍼지트

데코레이터

행위 패턴 (Behavoral)

옵서버

스테이트

스트래이지

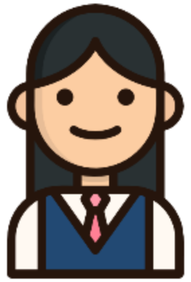
탬플릿 메서드

커맨드

02 옵저버(Observer) 패턴

한 객체의 상태 변화에 따라 다른 객체도 연동 되도록 일대다 객체 의존 관계구성하는 패턴

= 한 객체가 변동되면 관련된 객체들에게 알림이 감!



수강 신청 정정

02 옵저버(Observer) 패턴

예시

학생이 수강신청을 정정 했을 경우

```
public interface Observer {
    void update(String courseName);
}
```

02 옵저버(Observer) 패턴

예시

학생이 수강신청을 정정 했을 경우

```java
public class Professor implements Observer {
    private String name;

    public Professor(String name) {
        this.name = name;
    }

    @Override
    public void update(String courseName) {
        System.out.println("교수명 : " + name + ": 학생이 변경한 강의 " + courseName);
    }
}
```

```java
public class Student implements Observer {
    private String name;
    private StudentManagementSystem studentManagementSystem;

    public Student(String name, StudentManagementSystem studentManagementSystem) {
        this.name = name;
        this.studentManagementSystem = studentManagementSystem;
        studentManagementSystem.addObserver(this); // 학생이 자기 자신을 옵저버로 등록
    }

    public void registerCourse(String courseName) {
        System.out.println("이름: " + name + " 등록 강의 : " + courseName);
        // 학생이 수강 과목을 등록하면 옵저버에게 알림을 보냄
        studentManagementSystem.notifyObservers(courseName);
    }

    public void dropCourse(String courseName) {
        System.out.println("이름: " + name + " 삭제 강의 " + courseName);
        // 학생이 수강 과목을 삭제하면 옵저버에게 알림을 보냄
        studentManagementSystem.notifyObservers(courseName);
    }

    @Override
    public void update(String courseName) {
        System.out.println("이름: " + name + ": 수정된 강의: " + courseName);
    }
}
```

02 옵저버(Observer) 패턴

예시

학생이 수강신청을 정정 했을 경우

```java
import java.util.ArrayList;
import java.util.List;

public class StudentManagementSystem {
    private List<Observer> observers = new ArrayList<>();

    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    public void notifyObservers(String courseName) {
        for (Observer observer : observers) {
            observer.update(courseName);
        }
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        StudentManagementSystem system = new StudentManagementSystem();

        Professor professor1 = new Professor("John");
        Professor professor2 = new Professor("Alice");

        system.addObserver(professor1);
        system.addObserver(professor2);

        Student student1 = new Student("Bob", system);
        Student student2 = new Student("Eva", system);

        student1.registerCourse("Math");
        student2.registerCourse("History");

        student1.dropCourse("Math");
    }
}
```