# 11월 27일 (토)

## 1. JavaScript Decorator 디자인 패턴

### 데코레이터란?

데코레이터 패턴은 하나의 객체의 행위를 동적으로 추가, 수정 한다.

새로운 행위를 추가하는 것은 컴파일 시점이 아니라 런타임 시점에 추가가 된다. 수 많은 데코레이터들은 본 객체에 기능을 추가하거나 덮어쓸 수 있다.

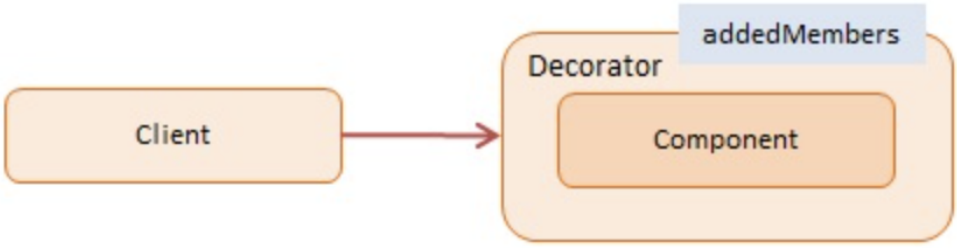
### 데코레이터의 사용

데코레이터의 하나의 예시 중 하나는 계층에 따른 보안관리이다. 예를 들어 HR 매니저는

데코레이터의 예로는 인증된 사용자의 권한에 따라 비즈니스 객체에 권한 있는 정보에 대한 추가 액세스 권한이 부여되는 보안 관리가 있습니다. 예를 들어, HR 관리자는 급여 정보를 볼 수 있도록 직원의 급여 레코드가 추가된(즉, 장식된) 직원 개체로 작업합니다.

데코레이터는 컴파일 시간에 발생하는 상속과 반대로 런타임 변경을 허용하여 정적으로 유형이 지정된 언어에 유연성을 제공합니다. 그러나 JavaScript는 동적 언어이며 런타임에 개체를 확장하는 기능은 언어 자체에 적용됩니다.

이러한 이유로 Decorator 패턴은 JavaScript 개발자와 관련이 적습니다. JavaScript에서 Extend 및 Mixin 패턴은 Decorator 패턴을 포함합니다. Dofactory JS 에서 이러한 패턴과 기타 최신 JavaScript 패턴에 대해 자세히 알아볼 수 있습니다 .




## 데코레이터 참여 객체

이 패턴에 참여하는 객체는 다음과 같습니다.

- **클라이언트**

  -- 예제 코드에서: run() 함수

  - 데코레이팅된 컴포넌트에 대한 참조를 유지합니다.
- **구성 요소사용자**

  -- 예제 코드에서:

  - 추가 기능이 추가된 개체
- **데코레이터DecoratedUser**

  -- 예제 코드에서:

  - 구성 요소에 대한 참조를 유지하여 구성 요소 '감싸기'
  - Component의 인터페이스를 준수하는 인터페이스를 정의합니다.
  - 추가 기능 구현(다이어그램의 addedMembers)

### 예시

예제 코드에서 `User` 개체는 `DecoratedUser` 개체에 의해 장식(향상)됩니다 . 여러 주소 기반 속성으로 사용자를 확장합니다. 원래 인터페이스는 동일하게 유지되어야 하며, 이는 `user.name` 에 할당된 이유를 설명합니다 `this.name` . 또한 `say` DecoratedUser 의 `say` 메소드는 User 의 메소드를 숨깁니다 .

JavaScript 자체는 추가 데이터 및 동작으로 객체를 동적으로 확장하는 데 훨씬 더 효과적입니다. Dofactory JS 에서 런타임 시 개체 확장 및 기타 기술에 대해 자세히 알아볼 수 있습니다 .

```
var User = function (name) {
    this.name = name;

    this.say = function () {
        console.log("User: " + this.name);
    };
}

var DecoratedUser = function (user, street, city) {
    this.user = user;
    this.name = user.name;  // ensures interface stays the same
    this.street = street;
    this.city = city;

    this.say = function () {
        console.log("Decorated User: " + this.name + ", " +
            this.street + ", " + this.city);
    };
}

function run() {

    var user = new User("Kelly");
    user.say();

    var decorated = new DecoratedUser(user, "Broadway", "New York");
    decorated.say();
}
```