Design Patterns

Factory Method (Creational)

팩토리 메소드 패턴 - 팩토리 메소드 패턴에서는 객체를 생성하기 위한 인터페이스를 정의하는데, 어떤 클래스 인스턴스를 만들지는 서브클래스에서 결정하게 만듭니다. 팩토리 메소드 패턴을 이용하면 클래스의 인스턴스를 만드는 일을 서브클래스에게 맡기는 것이죠.

Abstract Factory (Creational)

추상 팩토리 패턴 - 추상 팩토리 패턴에서는 인터페이스를 이용하여 서로 연관된, 또는 의존하는 객체를 구상 클래스를 지정하지 않고도 생성할 수 있습니다.

Adapter (Structural)

어댑터 패턴(Adapter Pattern) - 한 클래스의 인터페이스를 클라이언트에서 사용하고자 하는 다른 인터페이스로 변환합니다. 어댑터를 이용하면 인터페이스 호환성 문제 때문에 같이 쓸 수 없는 클래스들을 연결해서 쓸 수 있습니다.

Command (Behavioral)

커맨드 패턴 - 커맨드 패턴을 이용하면 요구 사항을 객체로 캡슐화 할 수 있으며, 매개변수를 써서 여러 가지 다른 요구 사항을 집어넣을 수도 있습니다. 또한 요청 내역을 큐에 저장하거나 로그로 기록할 수도 있으며, 작업취소 기능도 지원 가능합니다.

Composite (Structural)

컴포지트 패턴을 이용하면 객체들을 트리 구조로 구성하여 부분과 전체를 나타내는 계층구조로 만들 수 있습니다. 이 패턴을 이용하면 클라이언트에서 개별 객체와 다른 객체들로 구성된 복합 객체(composite)를 똑같은 방법으로 다룰 수 있습니다.

Decorator (Structural)

데코레이터 패턴에서는 객체에 추가적인 요건을 동적으로 첨가한다. 데코레이터는 서브클래스를 만드는 것을 통해서 기능을 유연하게 확장할 수 있는 방법을 제공한다.

Facade (Structural)

퍼사드 패턴 - 어떤 서브시스템의 일련의 인터페이스에 대한 통합된 인터페이스를 제공합니다. 퍼사드에서 고수준 인터페이스를 정의하기 때문에 서브시스템을 더 쉽게 사용할 수 있습니다.

Iterator (Behavioral)

이터레이터 패턴은 컬렉션 구현 방법을 노출시키지 않으면서도 그 집합체 안에 들어있는 모든 항목에 접근할 수 있게 해 주는 방법을 제공해 줍니다.

Observer (Behavioral)

옵저버 패턴(Observer Pattern)에서는 한 객체의 상태가 바뀌면 그 객체에 의존하는 다른 객체들한테 연락이 가고 자동으로 내용이 갱신되는 방식으로 일대다(one-to-many) 의존성을 정의합니다.

Singleton (Creational)

싱글턴 패턴 - 싱글턴 패턴은 해당 클래스의 인스턴스가 하나만 만들어지고, 어디서든지 그 인스턴스에 접근할 수 있도록 하기 위한 패턴입니다.

Proxy (Structural)

프록시 패턴 - 어떤 객체에 대한 접근을 제어하기 위한 용도로 대리인이나 대변인에 해당하는 객체를 제공하는 패턴

Strategy (Behavioral)

스트래티지 패턴(Strategy Pattern)에서는 알고리즘군을 정의하고 각각을 캡슐화하여 교환해서 사용할 수 있도록 만든다. 스트래티지를 활용하면 알고리즘을 사용하는 클라이언트와는 독립적으로 알고리즘을 변경할 수 있다.

State (Behavioral)

스테이트 패턴을 이용하면 객체의 내부 상태가 바뀜에 따라서 객체의 행동을 바꿀 수 있습니다. 마치 객체의 클래스가 바뀌는 것과 같은 결과를 얻을 수 있습니다.

Template Method (Behavioral)

템플릿 메소드 패턴에서는 메소드에서 알고리즘의 골격을 정의합니다. 알고리즘의 여러 단계 중 일부는 서브클래스에서 구현할 수 있습니다. 템플릿 메소드를 이용하면 알고리즘의 구조는 그대로 유지하면서 서브클래스에서 특정 단계를 재정의할 수 있습니다.

- 생성(Creational) 패턴
  - 추상 팩토리 (Abstract Factory)
  - 빌더 (Builder)
  - 팩토리 메서드 (Factory Methods)
  - 프로토타입 (Prototype)
  - 싱글턴 (Singleton)
- 구조(Structural) 패턴
  - 어댑터 (Adapter)
  - 브릿지 (Bridge)
  - 컴포지트 (Composite)
  - 데코레이터 (Decorator)
  - 퍼사드 (Facade)
  - 플라이웨이트 (Flyweight)
  - 프록시 (Proxy)
- 행위(Behavioral) 패턴
  - 책임 연쇄(Chain of Responsibility)
  - 커맨드 (Command)
  - 인터프리터 (Interpreter)
  - 이터레이터 (Iterator)
  - 미디에이터 (Mediator)
  - 메멘토 (Memento)
  - 옵저버 (Observer)
  - 스테이트 (State)
  - 스트래티지 (Strategy)
  - 템플릿 메서드 (Template Method)
  - 비지터 (Visitor)

MVC

- Model; Observer Pattern
- View; Composite Pattern
- Controller; Strategy Pattern

- 모델 뷰 컨트로러 패턴(MVC)은 옵저버 패턴, 스트래티지 패턴, 컴포지트 패턴으로 이루어진 컴파운드 패턴입니다.
- 모델에서는 옵저버 패턴을 이용하여 옵저버들에 대한 의존성은 없애면서도 옵저버들한테 자신의 상태가 변경되었음을 알릴 수 있습니다.

- 컨트롤러는 뷰에 대한 전략 객체입니다. 뷰에서는 컨트롤러를 바꾸기만 하면 다른 행동을 활용할 수 있습니다.
- 뷰에서는 컴포지트 패턴을 이용하여 사용자 인터페이스를 구현합니다. 보통 패널이나 프레임, 버튼과 같은 중첩된 구성요소로 구성됩니다.
- 모델과 뷰, 컨트롤러는 이 세 패턴을 통해서 서로 연결됩니다. 상대방에 대해 느슨하게 결합되기 때문에 깔끔하면서도 유연하게 구현할 수 있습니다.
- 새로운 모델을 기존의 뷰 및 컨트롤러하고 연결해서 쓸 때는 어댑터 패턴을 활용하면 됩니다.
- 모델 2는 MVC를 웹 애플리케이션에 맞게 적용한 디자인이라고 할 수 있습니다.
- 모델 2에서 컨트롤러는 서블릿으로 구현되며 뷰는 JSP/HTML로 구현됩니다.

java_spring_framework

From:
https://theta5912.net/ - **reth**

Permanent link:
https://theta5912.net/doku.php?id=public:computer:design_patterns&rev=1650006160

Last update: **2022/04/15 16:02**