|  |  |
| --- | --- |
| Observer Pattern | |
| Description | * 객체의 상태 변화를 관찰하는 관찰자들, 즉 observer들의 목록을 객체에 등록하여 상태 변화가 있을 때마다 method 등을 통해 객체가 직접 목록의 각 observer에게 통지 * 주로 분산 이벤트 핸들링 시스템을 구현하는데 사용 (발행/구독 모델) * 객체의 상태를 display하는 것과 객체 자체를 분리하여 다양한 display가 제공되도록 지원 * The Observer defines a one-to-many relationship so that when one object changes state, the others are notified and updated automatically. |
| Problem description | * 상태정보에 대해 다양한 방법으로 display하고자 할 때 * 적용되는 경우   (1) 상태 변화를 수신해야 하는 객체가 여러 개거나 전달받아야 할 정보가 많을 때  (2) 상태 정보가 한 가지 이상의 방법으로 display될 때  (3) 상태 정보를 유지하고 있는 객체는 어떤 방식으로 이 정보가 display되는지  알 필요 없게 할 때 |
| Solution description | * Two abstract objects : Subject, Observer * Two concrete objects : ConcreteSubject, ConcreteObserver * Subject가 Observer에 대해서 아는 것은 Observer가 Observer 인터페이스를 구현한다는 것 * Observer는 언제든지 새로 추가할 수 있음 (Subject는 Observer 인터페이스 구현하는 객체 목록에만 의존하기 때문) * 새로운 형식의 Observer를 추가하려 해도 Subject를 전혀 변경할 필요가 없음 (새로운 클래스에서 Observer 인터페이스만 구현하면 됨) * Subject나 Observer가 바뀌더라도 서로에게 영향을 주지 않음 ⇒ Subject와 Observer는 서로 독립적으로 재사용 가능 * 추상 객체는 모든 경우에 적용될 수 있는 일반적인 operations만을 포함 * ConcreteSubject는 Subject 클래스를 상속받아 정의되며, 화면에 보일 내용을 유지함 * Observer는 상태가 변화되었음을 알려 주는 역할을 담당 ⇒ Subject는Observer를 추가하거나 제거할 수 있음 * ConcreteObserver는 추상 객체인 Observer의 Update() method를 구현하여 ConcreteSubject의 상태를 복사하여 유지 * ConcreteObserver는 상태가 변화될 때마다 자동적으로 상태 변화를 화면에 보여 줌 |
| Consequences | Subject는 abstract observer에 대해서만 알고 있으며, concrete class에 대해서 알지 못함 ⇒ 해당 객체 간의 coupling 최소화 |

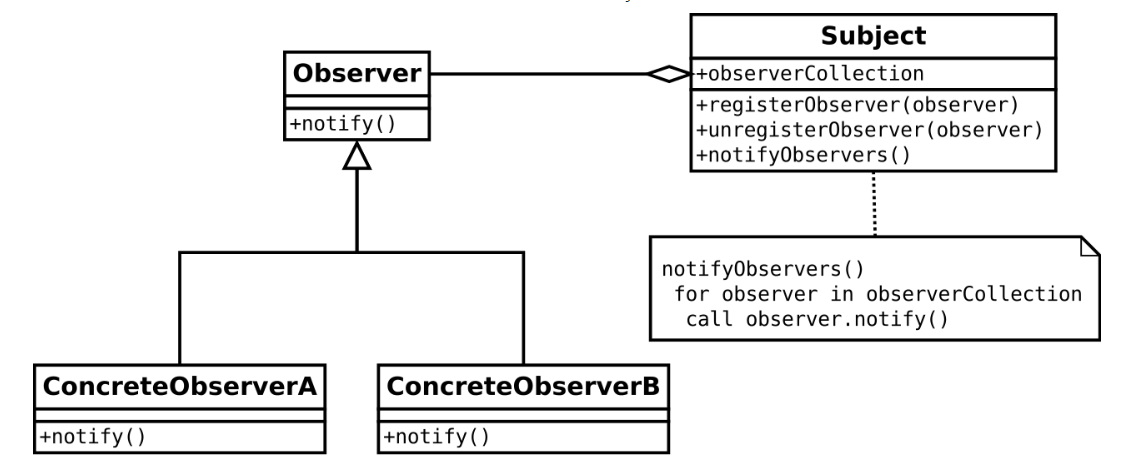


그림 1 Observer Pattern UML

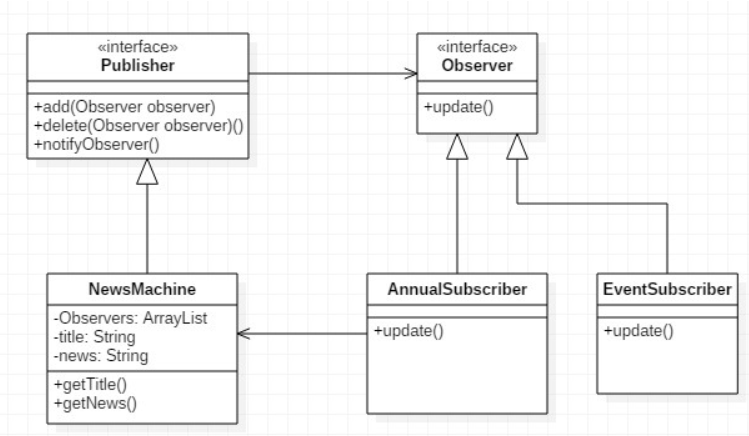


그림 2 Observer Pattern 예시 UML

|  |  |
| --- | --- |
| Façade Pattern | |
| Description | * Façade: 클래스 라이브러리 같은 어떤 소프트웨어의 다른 코드 부분에 대한 간략화된 Interface를 제공하는 객체 * 어떤 Sub-System의 일련의 Interface에 대한 통합된 Interface를 제공 * Façade에서 고수준 Interface를 정의하기 때문에 Sub-System을 더 쉽게 사용 가능 |
| Problem Description | * 여러 개로 분산되어 있어 복잡해 보이는 Class들을 하나로 편하게 보고 싶을 때 * 적용되는 경우 (1) 분산되어 있는 저차원 Class들을 효율적으로 관리해야 할 때 (2) 소프트웨어 라이브러리를 사용해야 하는 코드가 존재할 때 (3) API의 집합 작성이 복잡할 때 |
| Solution Description | * Sub-System Objects, Interface related with Sub-System Objects * Sub-System Objects에 사용되는 Interface의 공통된 Method들을 선별 * 선별한 Method들을 Façade Interface를 만들어 그곳에 모아 정리 * 어떠한 Method에서든지 아래와 같은 네 종류의 Objects의 Method만을 호출하게 해야 함 (1) Object 자체 (2) Method에 매개 변수로 전달된 Object (3) 해당 Method에서 생성하거나 Instance를 만든 Object (4) 해당 Object에 속하는 구성 요소 |
| Consequences | * Sub-System들을 Interface로 감싸 직접 처리해야 할 Objects의 수를 줄이고 Sub-System을 더욱 사용하기 쉽게 만들어 준다. * Sub-System과 그것의 Client의 강한 의존성을 약화시킨다. |

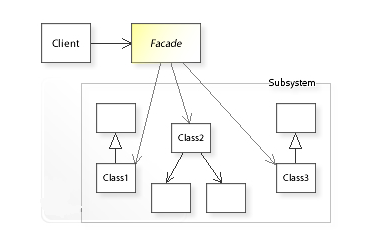


그림 Façade Pattern UML

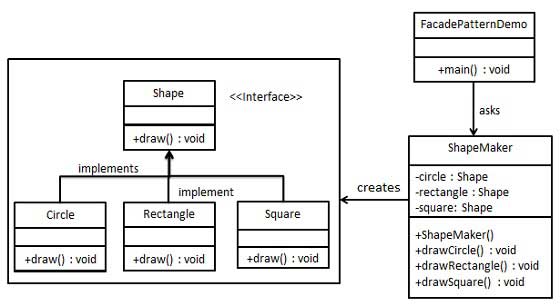


그림 4 Façade Pattern 예시 UML

|  |  |
| --- | --- |
| Iterator Pattern | |
| Description | 컬렉션 구현 방법을 노출시키지 않으면서도, 그 집합체 안에 들어 있는 모든 항목에 접근할 수 있게 해 주는 방법을 제공 |
| Problem Description | 여러 종류의 컬렉션 전체의 원소들에 대해 동일한 action을 적용할 때 사용 |
| Solution Description | * Iterator 인터페이스 : 원소를 순서대로 검색해 나감 * hasNext() : 다음 원소가 존재하는지 알려 줌 * next() : 다음 원소를 얻음 * ConcreteIterator 클래스 : Iterator 인터페이스를 implement함, 검색하기 위해 필요한 정보를 가지고 있어야 하며, 해당 자료구조나 집합체가 가진 모든 원소를 순회할 수 있는 가능 정의 * Aggregate 인터페이스 : Iterator를 생성하는 인터페이스, 선언된 method는 Iterator를 생성하는 것 하나뿐 * iterator() : 집합체에 대응하는 Iterator 인스턴스 1개 생성 * ConcreteAggregate 클래스 : Aggregate 인터페이스를 implement함, ConcreteIterator의 인스턴스 생성 * 사용할 집합체 안에 Iterator를 만드는 메소드를 제공해야 한다 |
| Consequences | 모든 집합체에 대하여 동일한 인터페이스로 모든 원소를 순회할 수 있도록 해 줌 |

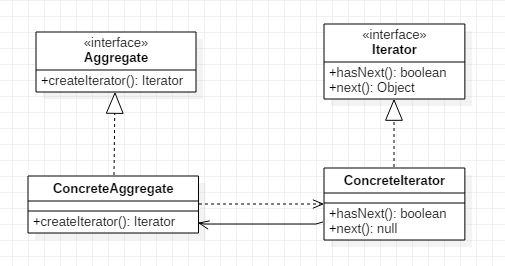


그림 5 Iterator Pattern UML

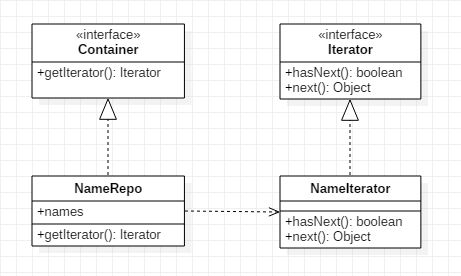


그림 6 Iterator Pattern 예시 UML

|  |  |
| --- | --- |
| Decorator Pattern | |
| Description | * The decorator pattern can be used to extend (decorate) the functionality of a certain object statically, or in some cases at run-time, independently of other instances of the same class, provided some groundwork is done at design time. * 데코레이터 패턴(Decorator pattern)이란 주어진 상황 및 용도에 따라 어떤 객체에 책임을 덧붙이는 패턴. * 기능 확장이 필요할 때 서브클래싱 대신 쓸 수 있음. |
| Problem Description | * 객체에 동적으로 새로운 책임을 추가하고 싶을 때. * 기능을 추가해야 할 때 |
| Solution Description | 1. Subclass the original Component class into a Decorator class (see UML diagram);  2. In the Decorator class, add a Component pointer as a field  3. In the Decorator class, pass a Component to the Decorator constructor to initialize the Component pointer;  4. In the Decorator class, forward all Component methods to the Component pointer.  5. In the ConcreteDecorator class, override any Component method(s) whose behavior needs to be modified. |
| Consequences | 객체에 추가적인 요건을 동적으로 추가할 수 있다. |



그림 Decorator Pattern UML

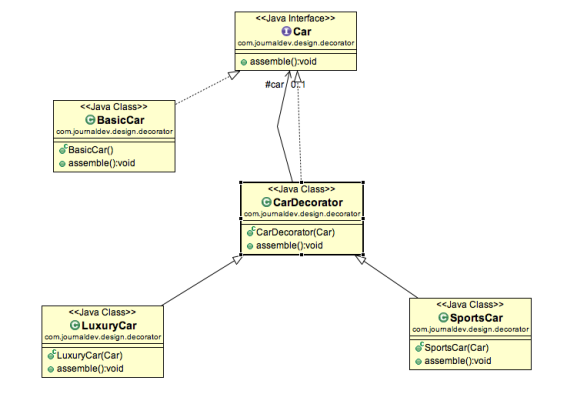


그림 8 Decorator Pattern 예시 UML