#define design pattern

20512 김택서

목차

디자인패턴의 정의

종류와 쓰임새

느낀 점

1.디자인패턴의 정의

찾아본 문서들 중 어느 것도 이러한 용어로 기술하지는 않았지만 굳이 한국어로 번역하여 **반복적 설계**라고 칭하고 싶다. 물론 단어 그 자체로는 용어의 뜻을 이해하기 어렵지만 그 뜻을 안 후에는 디자인패턴보다 조금 더 직관적인 표현이 아닐까 싶어서이다. 일단, 서술 하자면 반복적설계는 프로그래밍에서의 반복적인 생산활동에 따른 양식 또는 매뉴얼 같은 것이다. 예를 들어 C++에서의 같은 구조를 지니며 자료형만다른 형태를 띄는 반복적인 코드를 간결 화하고 추상적인 개념으로 표시하기위해 사용되는 템플릿문법처럼 여러 프로그램이나 프로젝트에서 사용되는 설계 형태를 일종의 틀처럼 **정형 시켜** 프로그래머의 생산성을 이끄는 용도이니 말이다. 코드의 반복 부분이 쓰임새가 다른 같은 형태의 반복적인 코드 인가와 같은 쓰임새의 다른 형태의 반복적인 프로젝트(코드의 상위 범주)인가의 차이점이 있기는 하지만 모두 추상적인 개념으로써 프로그래머의 생산성을 지향하고 **인력의 효율을 상향**시켜주는 부분에 있어서는 같은 뜻을 한다고 본다.

디자인 패턴의 사용 용도는 추상적인 방식으로 프로그램의 구조를 정형화 시키는 부분에서부터 나오는 생산력 향상에 관한 부분 뿐만이 아니라 협업 시간 단축 에 의한 장점도 지니고 있다. 협업 시간 단축이란 디자인 패턴을 사용할 경우 프로젝트의 개발자가 해당 내용을 숙지 시 향후 개발방향과 문제해결부분에 있어서의 **원활한 의사소통**으로 나오는 시간적 단축을 의미한다. 같은 맥락으로 이미 제작 되어있는 디자인 패턴을 사용하므로 프로젝트 제작 자체에 소비되는 시간 또한 줄일 수 있게 된다. 조금 더 나아가서는 프로젝트의 틀이 정해져 있는 것이므로 향후 프로그램의 기능을 추가, 또는 삭제 등 수정을 가하여야 하는 경우에는 **유연한 대처**가 가능해진다.

단점으로는 그 범용성과 초기 자원 **투자비용의 부담**이 있다. 장점들이 있듯이 디자인 패턴에 관한 단점도 마땅히 있어야 되겠다. 먼저 범용성이다. 대부분의 디자인패턴은 객체지향 프로그램의 설계나 구현에 많이 사용된다. 이는 표현 방식 상 객체지향에 알맞기 때문인데, 이러한 이유로 구조적 설계와 구현에서는 복잡하게 되어 잘 사용하지 않는다. 두번째는 초기 자원 투자 비용의 부담이다. 이는 UML제작에서와의 비슷한 이유에서인데 디자인 패턴을 사용함으로써 의 자원 절감도 분명히 있겠지만 무엇보다 디자인패턴을 사용하는 부분에서도 **자원이 소모**된다는 것이다. 때문에 개발자들은 프로젝트제작 부분에서 어떠한 방향이 가장 **효율**적인지를 고려해야한다.

코딩의 설계적인 부분에서의 기법들을 설명하는 서적 코드 패턴 그리고 소프트웨어의 구문을 인용하자면 *‘디자인 패턴(Design Pattern) 이란 방식을 통해 소프트웨어 설계에서 얻은 세세한 경험들을 기록해 놓도록 하는 것 패턴 "어떤 상황의 문제에 대한 해법" 일반적으로 하나의 패턴에는 다음의 네 가지 요소가 반드시 들어가 있다.’*라고 정의하였다.

다음의 네 요소란 이와 같다.

*1 패턴 이름(pattern name)은 한두 단어로 설계 문제와 해법을 서술한다.*

*2 문제(problem)는 언제 패턴을 사용하는가를 서술하며 해결할 문제와 그 배경을 설명한다.*

*3 해법(solution)은 설계를 구성하는 요소들과 그 요소들 간의 관계 책임 그리고 협력 관계를 서술한다.*

*4 결과(consequence)는 디자인 패턴을 적용해서 얻는 결과와 장단점을 서술한다*

각각의 요소에 대하여 하나씩 살펴보자면 첫번째는 다른 프로그래머와 소통하는 부분에 있어서의 중요성을 강조하고 있다. 그 이유는 다음과 같다. 디자인 패턴 뿐만이 아니라 다른 협업에서의 팀원과 **소통 및 협업 능력**은 개인의 역량보다도 중요하다고 판단할 수 있다. 이에 있어서 다른 프로그래머들과의 협업을 장점으로 가지는, 즉 디자인 패턴에 있어서의 핵심 부분이라고 할 수 있다. 때문에 각 프로그래머 사이에서의 작명 센스는 무엇보다도 중요한 부분인것이다. 이러한 센스는 디자인 패턴 뿐 만이 아니라 변수 명이나 함수 이름 등 코드의 가독성과 소프트웨어의 구조 파악에 중요한 역할을 미치는 요소에서는 필수적인 약속이라고 할 수 있다.

두번째와 세번째는 묶어서 설명할 것이다. 여기서의 문제(problem)와 해법(solution)은 설명서의 한 부분들과 같다. 예를 들어 한 제품의 설명서에는 이 제품의 사용 용도와 그 사용 방법을 기술하는 것이 일반적일 것이다.(해당 제품을 처음 접하는 사람들을 타겟으로 제작된 설명서를 기준) 여기서 문제는 사용용도, 해법은 사용방법에 해당된다. 각 디자인패턴은 고유한 성질을 지니고 있으며 그 종류에 따라 생김새도 역할도 사용하는 방법도 다양하다. 때문에 디자인패턴에 있어서 해당 디자인 패턴의 사용을 위해서는 그 사용 용도와 방법을 아는 것이 당연하다. 또한 디자인 패턴은 그 사용방법에 있어서 독이 될 수도 약이 될 수도 있기 때문에 해당 디자인패턴이 **사용될 용도를 제시**하는 것이 마땅하다고 볼 수 있다.

마지막으로 결과는 제품의 리뷰와 비슷하다. 다른 점이라면 판매자가 직접 리뷰를 작성한다는 것인데 디자인패턴에서의 단점에서 서술했듯이 디자인패턴을 사용하기 전에 그 효율성을 따지는 중요한 지표가 되기 때문이다. 제품을 사기전 그 제품을 사용해도 되는 것인지, 가장 효율적인 방안인지를 판단하는 중요한 **판단근거**로써 디자인패턴의 결과(consequence)는 그 역할을 다하여야 하는 것이다.

2.예1-추상 팩토리 패턴(abstract factory pattern)

디자인패턴이란 큰 개념을 이해하기위해 다양한 종류의 디자인패턴중 추상팩토리를 중심으로 설명할 것이다. 일단 추상 팩토리 패턴을 앞서 설명했던 네 가지의 요소에 따라 설명하기전에 추상 팩토리 패턴이 속한 범주인 생성 패턴에 대하여 알아보고 가는 것이 좋을 것 같다. 패턴의 목적에 따라 크게 세가지로 구별할 수 있는데 생성, 구조, 행동 이 그 목적들이다. 생성은 객체의 생성 과정에 관여하며 구조 패턴은 클래스나 객체의 합성에 관한 부문이며 행동 패턴은 클래스나 객체들이 상호작용하는 방법과 책임을 분산하는 방법을 정의한다. 다음은 네가지의 요소,에따른 설명이다

1. 패턴이름

직역하면 추상 공장, 추상공장 에서는 사용자에게 인터페이스(API)를 제공하고, 인터페이스만 사용해서 부품을 조립하여 만든다. 즉 추상적인 부품을 조합해서 추상적인 제품을 만든다.

1. 문제

추상 공장 디자인패턴은 객체 생성을 클라이언트가 직접 하는 것이 아니라, 간접적으로 수행함으로써 클라이언트가 객체의 생성이나 구성 또는 표현 방식을 독립적이도록 만들고자 할 때

여러 제품 군 중 사용 제품 군을 쉽게 선택할 수 있도록 만들고 싶을 때

서로 관련된 제품들이 하나의 제품 군을 형성하고, 이런 제품 군이 여러 개 존재하는 상황에서 생성되는 제품 객체가 항상 같은 제품 군에 속하는 것을 확실히 보장하고 싶을 때

제품들에 대한 클래스 라이브러리를 만들어야 하는데 그 인터페이스만 드러내고 구현은 숨기고 싶을 때 등에 사용할 수 있다

1. 해법

해당 디자인 패턴을 어떻게 사용하는지에 관한 부분은 뒤에 예제를 이용한 설명에서 하도록 하겠다.

1. 결과

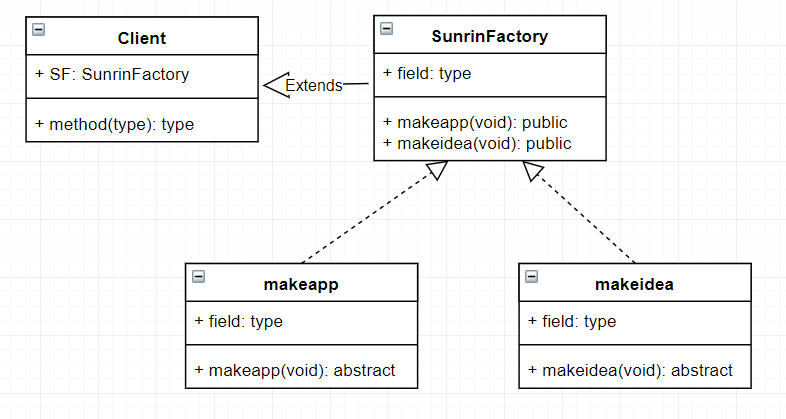
장점으로는 객체가 생성되는 방식이나 과정 및 책임을 클라이언트가 관여하지 않아도 된다는 점이다 또한 제품 군간 교체가 쉽고 제품 군들이 실수로 섞여서 사용되는 것 자연스럽게 방지할 수 있다는 것이 장점이다

단점에는 제품 군의 개수가 늘어날수록 클래스의 개수도 늘어나야 한다는 점과 새로운 제품이 추가될 경우 기존의 모든 팩토리 클래스를 수정해야 한다는 점이 있다.

예시코드

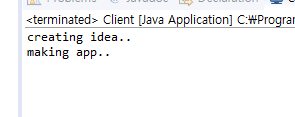
|  |
| --- |
| AppFactory.java(interface) |
| **package** designpattern;  **public** **interface** AppFactory {  **public** **void** makeapp();  } |
| IdeaFactory.java(interface) |
| **package** designpattern;  **public** **interface** IdeaFactory {  **public** **void** makeidea();  } |
| SunrinFactory.java(class) |
| **package** designpattern;  **public** **class** SunrinFactory **implements** IdeaFactory , AppFactory{  @Override  **public** **void** makeapp() {  // **TODO** Auto-generated method stub  System.***out***.println("making app..");  }  @Override  **public** **void** makeidea() {  // **TODO** Auto-generated method stub  System.***out***.println("creating idea..");  }  } |
| Client.java(class) |
| **package** designpattern;  **public** **class** Client **extends** SunrinFactory{  **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  SunrinFactory SF = **new** SunrinFactory();  SF.makeidea();  SF.makeapp();  }  } |

위는 선린 팩토리라는 가상의 공장에서 만들어지는 아이디어와 앱을 클라이언트에서 실행하는 abstract factory이다



이처럼 가상 팩토리는 서로 관련성이 있거나 독립적인 여러 객체의 집합을 생성할 수 있는 인터페이스 제공하는 패턴으로써 사용자가 제품 내부의 수행방식을 알지 않아도 되게할 수 있도록 만들어주는 디자인 패턴이다

<실행결과>



4.느낀 점

디자인패턴과 UML을 독립적인 방식으로 구별

때문에 디자인 패턴이라는 개념의 이해와 디자인패턴을 성형화 하는 것에 어려움을 겪었음

초기에는 UML과의 연관성이 생각나서 디자인 패턴과 UML의 관계와 차이 점등을 해당 보고서의 큰 갈래로 잡았으나 후에 디자인 패턴의 개념을 점차 이해하게 되며 보고서의 주제를 바꾸게 됨

생성 패턴과 구조 패턴 행동패턴 등에 대해서 보고서에 담지 못한 것이 아쉽다

여러 종류의 디자인패턴을 모두 비교하지 못하여 예시로 삼을 디자인패턴을 고르는 것이 어려웠고 그로 인해 자연스러운 인과관계를 가진 보고서를 완성하지 못한 것이 부족했다.

보고서를 제출하는 것이 목적이 아닌 직접 알고 작성과 정리하는 것에 목적을 두어 생각보다 시간이 오래 걸린 점에서 시간 배분과 스케줄관리의 필요성을 느낌

참고문헌

<http://ohgyun.com/279>

<https://wikidocs.net/book/55>

<http://hyeonstorage.tistory.com/99>

<http://hyeonstorage.tistory.com/165>

<http://blog.chakannom.com/2015/11/design-pattern_11.html>

<http://blog.naver.com/writer0713/221087123896>

<http://terms.naver.com/entry.nhn?docId=3532957&cid=58528&categoryId=58528>

<http://hyeonstorage.tistory.com/146>

<https://wikidocs.net/898>

<http://jdm.kr/blog/192>

<http://blog.chakannom.com/2015/11/abstract-factory-pattern.html>

<http://terms.naver.com/entry.nhn?docId=3532963&cid=58528&categoryId=58528&expCategoryId=58528>

<http://jwprogramming.tistory.com/68>