**[ 13 ] 패턴**

1. 패턴(디자인 패턴)이란?

객체지향이 세상에 등장한 이후로 훌륭한 우리의 선배 개발자들은 어떻게 하면 더욱 빠르게 개발하고, 유지 보수 측면에서도 효율적으로 개발을 할 수 있을까에 대한 고민을 많이 해왔습니다.

(정상적인 개발자들은 기간 내 프로젝트를 빨리 끝낼지와 함께 계속 유지보수측면에서 유지보수 하기 쉽게 개발 초기단계부터 깔끔하게 개발할 수 있을까를 고민합니다. 여기서 “유지보수 하기 쉽게 깔끔하게”가 중요합니다)

사실 지금도 하고 있습니다. 이러한 고민을 하던 중 객체지향 언어의 상속, 인터페이스, 추상화 등의 기능을 이용해서 시간과 인력 투입의 효율을 극대화 하기 위한 방식이 정립되기 시작했습니다.

(선배와 후배와 동료들과 개발할 때 어떤 흐름에 의해 코딩을 하더라는 것입니다. 그 흐름 어떤 패턴을 가지고 코딩을 하는 것을 하나씩 하나씩 정립화 시켜 놓은 것이 바로 패턴입니다)

그리고 이러한 방식을 어떠한 틀로 정해 놓은 것이 바로 디자인 패턴 입니다.

쉽게 정리하면, **우리의 개발자 선배님들이 객체지향 언어의 장점들을 모아 가장 효율적으로 개발을 할 수 있게 만들어 놓은 프레임**입니다.

2. 싱클톤 패턴 : 싱글톤 패턴(Singleton pattern)이란 어떤 클래스의 객체는 오직 하나인 유일한 객체를 만들어 여러가지 상황에서 동일한 객체에 접근하기 위해 만들어진 패턴이다.

(하나의 클래스를 통해 obj1, obj2, obj3이라는 객체 생성 시, obj2와 obj3는 전혀 다른 독립체. obj2.i와 obj3.i는 별개였지만,

cf. 어떤 클래스는 객체를 유일하게 하나만 만들 수 있다. 어떻게 하나만 만들 수 있는지 알아본다)

위키백과 曰, 싱글턴 패턴(Singleton pattern)을 따르는 클래스는, 생성자가 여러 차례 호출되더라도 실제로 생성되는 객체는 하나이고 이 객체에 접근할 수 있는 **전역적인 접촉점**을 제공하는 패턴이다.

지금까지의 클래스를 잘 생각해 보면, 클래스를 하나 만들고, 만들어진 클래스로부터 객체를 여러 개 생성하여 사용하였습니다. 하지만 싱글턴 패턴을 이용하면, 클래스 하나에 객체를 유일하게 하나만 생성하여 모든 곳에서 하나의 객체에만 접근할 수 있습니다. 즉 전역의 개념으로 객체를 사용할 수 있습니다.

<싱글톤예제>

**public** **class** SingletonClass {

**private** **static** SingletonClass *SINGLETON\_INSTANCE*;

**private** **int** i;

**private** SingletonClass(){i = 10;} //생성자함수가 private이므로 외부에서는 new를 이용해서

// 생성못하고, 이 클래스 내부에서만 생성자함수 호출할 수 있다.

**public** **static** SingletonClass getSingletonClass(){

// 객체가 생성되기 전에 데이터영역의 클래스 상태에서 바로 접근가능한 메소드

**if**(*SINGLETON\_INSTANCE*==**null**)

*SINGLETON\_INSTANCE* = **new** SingletonClass();//객체생성을 이곳에서만 한다

**return** *SINGLETON\_INSTANCE*;

}

**public** **int** getI() {**return** i;}

**public** **void** setI(**int** i) {**this**.i = i;}

}

**public** **class** FirstClass {

**public** FirstClass(){

SingletonClass singletonObject = SingletonClass.*getSingletonClass*();

System.***out***.println("FirstClass 객체 ");

System.***out***.println(singletonObject.getI());

singletonObject.setI(999);

System.***out***.println("i = "+singletonObject.getI());

}

}

**public** **class** SecondClass {

**public** SecondClass() {

SingletonClass singletonObject = SingletonClass.*getSingletonClass*();

System.***out***.println("SecondClass 객체 ");

System.***out***.println(singletonObject.getI());

}

}

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

FirstClass firstObj = **new** FirstClass();

SecondClass secondObj = **new** SecondClass();

SingletonClass singObj = SingletonClass.*getSingletonClass*();

System.***out***.println("main()메소드에서 singleton객체");

System.***out***.println(singObj.getI());

}

}