**[ 12 ] 패턴**

목표 : Singleton pattern, Strategy pattern을 이해한다

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 끝낼 지와 함께 계속 유지보수측면에서 유지보수 하기 쉽게 개발 초기단계부터 깔끔하게 개발할 수 있을까를 고민합니다. 여기서 “유지보수 하기 쉽게 깔끔하게”가 중요합니다)

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 객체**를 만들어 여러가지 상황에서 동일한 객체에 접근하기 위해 만들어진 패턴.

(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());

}

}

3. 스트레티지 패턴(Strategy Pattern)

기능 하나를 정의하고 각각을 캡슐화하여 교환해서 사용할 수 있도록 만든다.

스트래티지를 활용하면 기능(알고리즘)을 사용하는 클라이언트와는 독립적으로 기능(알고리즘)을 변경할 수 있습니다..

쉽게 말하면, 어떤 객체를 만들 때 객체가 가지는 기능들이 다양하게 존재할 것입니다. 이러한 기능들을 추상화하여 언제든지 적용할 수 있게 만드는 것입니다. 즉 기능을 부품화, 표준화 하는 것입니다.

<ex 1>

시나리오1

모든 로봇은 기본적으로 걷고, 달릴 수 있어야 합니다.

로봇 모양은 팔, 다리, 머리, 몸통으로 이루어져 있습니다.

Super 로봇 : 날 수 O. 미사일을 쏠 수 O. 레이저검.

Standard로봇: 날 수 X. 미사일을 쏠 수 O. 목검

Low 로봇 : 날 수 X. 미사일을 쏠 수 X. 검 없음.

위 3가지 형태의 로봇을 만들어 보도록 하고, 추후 로봇들은 새로 만들어질 수도 있고, 업그레이드 되어질 경우를 생각하여 유연한 프로그램을 만들어 보도록 합니다.

1단계 : 각각 SuperRobot, StandardRobot, LowRobot 클래스 정의하고 객체 만들어 쓴다. 세 클래스의 중복된 코드가 많다. 그래서 2단계

**public** **class** SuperRobot {

**public** SuperRobot() { }

**public** **void** actionWalk(){System.***out***.println("걸을 수 있습니다");}

**public** **void** actionRun(){System.***out***.println("달릴 수 있습니다");}

**public** **void** shape(){System.***out***.println("슈퍼로봇입니다. 외형은 팔 다리 몸통 머리가 있습니다");}

**public** **void** actionFly(){System.***out***.println("날 수 있습니다");}

**public** **void** actionMissile(){System.***out***.println("마사일을 쏠 수 있습니다");}

**public** **void** actionknife() {System.***out***.println("레이저 검이 있습니다");}

}

**public** **class** StandardRobot {

**public** StandardRobot() { }

**public** **void** actionWalk(){System.***out***.println("걸을 수 있습니다");}

**public** **void** actionRun(){System.***out***.println("달릴 수 있습니다");}

**public** **void** shape(){System.***out***.println("스탠다드 로봇입니다. 외형은 팔 다리 몸통 머리가 있습니다");}

**public** **void** actionFly(){System.***out***.println("날 수 없습니다");}

**public** **void** actionMissile(){System.***out***.println("마사일을 쏠 수 있습니다");}

**public** **void** actionknife() {System.***out***.println("목검이 있습니다");}

}

**public** **class** LowRobot {

**public** LowRobot() { }

**public** **void** actionWalk(){System.***out***.println("걸을 수 있습니다");}

**public** **void** actionRun(){System.***out***.println("달릴 수 있습니다");}

**public** **void** shape(){System.***out***.println("로우 로봇입니다. 외형은 팔 다리 몸통 머리가 있습니다");}

**public** **void** actionFly(){System.***out***.println("날 수 없습니다");}

**public** **void** actionMissile(){System.***out***.println("마사일을 쏠 수 없습니다");}

**public** **void** actionknife() {System.***out***.println("검이 없습니다");}

}

**public** **class** RobotTestMain {

**public** **class** RobotMain {

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

SuperRobot superR = **new** SuperRobot();

StandardRobot standarR = **new** StandardRobot();

LowRobot lowR = **new** LowRobot();

System.***out***.println("---------------------------------");

superR.shape();

superR.actionWalk();

superR.actionRun();

superR.actionFly();

superR.actionMissile();

superR.actionknife();

System.***out***.println("---------------------------------");

standarR.shape();

standarR.actionWalk();

standarR.actionRun();

standarR.actionFly();

standarR.actionMissile();

standarR.actionknife();

System.***out***.println("---------------------------------");

lowR.shape();

lowR.actionWalk();

lowR.actionRun();

lowR.actionFly();

lowR.actionMissile();

lowR.actionknife();

}

}//class

2단계 : SuperRobot, StandardRobot, LowRobot 클래스의 공통점을 갖은 슈퍼클래스 만들고 SuperRobot, StandardRobot, LowRobot은 상속시키고 공통점은 뺀다.

**public** **class** Robot {

**public** **void** actionWalk(){System.***out***.println("걸을 수 있습니다");}

**public** **void** actionRun(){System.***out***.println("달릴 수 있습니다");}

**public** **void** shape(){System.***out***.println(getClass().getName() +"형 로봇입니다. 외형은 팔 다리 몸통 머리가 있습니다");}

}

**public** **class** SuperRobot **extends** Robot{

**public** SuperRobot() { }

**public** **void** actionFly(){System.***out***.println("날 수 있습니다");}

**public** **void** actionMissile(){System.***out***.println("마사일을 쏠 수 있습니다");}

**public** **void** actionknife() {System.***out***.println("레이저 검이 있습니다");}

}

**public** **class** StandardRobot **extends** Robot{

**public** StandardRobot() { }

**public** **void** actionFly(){System.***out***.println("날 수 없습니다");}

**public** **void** actionMissile(){System.***out***.println("마사일을 쏠 수 있습니다");}

**public** **void** actionknife() {System.***out***.println("목검이 있습니다");}

}

**public** **class** LowRobot **extends** Robot{

**public** LowRobot() { }

**public** **void** actionFly(){System.***out***.println("날 수 없습니다");}

**public** **void** actionMissile(){System.***out***.println("마사일을 쏠 수 없습니다");}

**public** **void** actionknife() {System.***out***.println("검이 없습니다");}

}

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

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

Robot[] robots = {**new** SuperRobot(),**new** StandardRobot(),**new** LowRobot()};

**for**(Robot r :robots){

r.shape(); r.actionWalk(); r.actionRun();

**if**(r **instanceof** SuperRobot){

SuperRobot sur = (SuperRobot) r;

sur.actionFly(); sur.actionMissile(); sur.actionknife();

}**else** **if**(r **instanceof** StandardRobot){

StandardRobot str =(StandardRobot) r;

str.actionFly();str.actionMissile();str.actionknife();

}**else** **if**(r **instanceof** LowRobot){

LowRobot str =(LowRobot) r;

str.actionFly();str.actionMissile();str.actionknife();

}//if

}//for

}//main

}//class

3단계 : 추상클래스로 추상화 한다. 각각 SuperRobot, StandardRobot, LowRobot 클래스의 actionFly(), actionMissile(), actionKnife()메소드는 있고 안의 구현하는 부분만 다르다. 추상화 시키면 다른 사람이 와서 다른 종류의 로봇을 만든다고 해도 actionFly(), actionMissile(), actionKnife() 은 빼지 않고 넣게 된다. 혹시 사용자가 ARobot형을 추가 시킨다고 해도 Robot에서 상속받아 쓴다면 반드시 actionFly(), actionMissile(), actionKnife()은 구현하게 된다

**public** **abstract** **class** Robot {

**public** **void** actionWalk(){System.***out***.println("걸을 수 있습니다");}

**public** **void** actionRun(){System.***out***.println("달릴 수 있습니다");}

**public** **void** shape(){System.***out***.println(getClass().getName() +"형 로봇입니다. 외형은 팔 다리 몸통 머리가 있습니다");}

**public** **abstract** **void** actionFly();

**public** **abstract** **void** actionMissile();

**public** **abstract** **void** actionknife();

}

**public** **class** SuperRobot **extends** Robot{

**public** SuperRobot() { }

@Override

**public** **void** actionFly(){System.***out***.println("날 수 있습니다");}

@Override

**public** **void** actionMissile(){System.***out***.println("마사일을 쏠 수 있습니다");}

@Override

**public** **void** actionknife() {System.***out***.println("레이저 검이 있습니다");}

}

**public** **class** StandardRobot **extends** Robot{

**public** StandardRobot() { }

@Override

**public** **void** actionFly(){System.***out***.println("날 수 없습니다");}

@Override

**public** **void** actionMissile(){System.***out***.println("마사일을 쏠 수 있습니다");}

@Override

**public** **void** actionknife() {System.***out***.println("목검이 있습니다");}

}

**public** **class** LowRobot **extends** Robot{

**public** LowRobot() { }

@Override

**public** **void** actionFly(){System.***out***.println("날 수 없습니다");}

@Override

**public** **void** actionMissile(){System.***out***.println("마사일을 쏠 수 없습니다");}

@Override

**public** **void** actionknife() {System.***out***.println("검이 없습니다");}

}

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

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

Robot[] robots = {**new** SuperRobot(),**new** StandardRobot(),**new** LowRobot()};

**for**(Robot r :robots){

r.shape(); r.actionWalk(); r.actionRun();

r.actionFly(); r.actionMissile(); r.actionknife();

}//for

}//main

}//class

4단계 : 3단계에서 좀 더 유연하여, 좀 더 유지보수도 쉽고 추후 업데이트도 쉽게 만들 수 있도록 리펙토링한다 ex.현재는 날수있다없다 에서 저공비행을 한다 고공비행을 한다는 좀더 업데이트된 기능으로 업데이트 할 수 있다고 생각하자

모든 기능을 객체화(부품화) 하였다. // object modularization

**public** **interface** IFly {

**public** **void** fly();

}

**public** **class** FlyNo **implements** IFly {

@Override

**public** **void** fly() {

System.***out***.println("날 수 없습니다");

}

}

**public** **class** FlyYes **implements** IFly {

@Override

**public** **void** fly() {

System.***out***.println("날 수 있습니다");

}

}

**public** **interface** IMissile {

**public** **void** missile();

}

**public** **class** MissileNo **implements** IMissile {

@Override

**public** **void** missile() {

System.***out***.println("미사일을 쏠 수 없습니다.");

}

}

**public** **class** MissileYes **implements** IMissile {

@Override

**public** **void** missile() {

System.***out***.println("미사일을 쏠 수 있습니다.");

}

}

**public** **interface** IKnife {

**public** **void** knife();

}

**public** **class** KnifeLazer **implements** IKnife {

@Override

**public** **void** knife() {

System.***out***.println("레이저검이 있습니다.");

}

}

**public** **class** KnifeNo **implements** IKnife {

@Override

**public** **void** knife() {

System.***out***.println("검이 없습니다.");

}

}

**public** **class** KnifeWood **implements** IKnife {

@Override

**public** **void** knife() {

System.***out***.println("목검이 있습니다.");

}

}

**public** **abstract** **class** Robot {

**public** **void** actionWalk(){System.***out***.println("걸을 수 있습니다");}

**public** **void** actionRun(){System.***out***.println("달릴 수 있습니다");}

**public** **void** shape(){System.***out***.println(getClass().getName() +"형 로봇입니다. 외형은 팔 다리 몸통 머리가 있습니다");}

**public** **abstract** **void** actionFly();

**public** **abstract** **void** actionMissile();

**public** **abstract** **void** actionknife();

}

**package** com.ch.ex4robot.modularization;

**import** com.ch.robot.inter.\*;

**public** **class** SuperRobot **extends** Robot{

/\*private IFly fly = new FlyYes();

private IMissile missile = new MissileYes();

private IKnife knife = new KnifeLazer();\*/

**private** IFly fly;

**private** IMissile missile;

**private** IKnife knife;

**public** SuperRobot() {

/\*fly = new FlyYes();

missile = new MissileYes();

knife = new KnifeLazer();\*/

setFly(**new** FlyYes());

setMissile(**new** MissileYes());

setKnife(**new** KnifeLazer());

}

@Override

**public** **void** actionFly(){fly.fly();}

@Override

**public** **void** actionMissile(){missile.missile();}

@Override

**public** **void** actionknife() {knife.knife();}

**public** **void** setFly(IFly fly) {**this**.fly = fly;}

**public** **void** setMissile(IMissile missile) {**this**.missile = missile;}

**public** **void** setKnife(IKnife knife) {**this**.knife = knife;}

}

**public** **class** StandardRobot **extends** Robot{

**private** IFly fly;

**private** IMissile missile;

**private** IKnife knife;

**public** StandardRobot() {

setFly(**new** FlyNo());

setMissile(**new** MissileYes());

setKnife(**new** KnifeWood());

}

@Override

**public** **void** actionFly(){fly.fly();}

@Override

**public** **void** actionMissile(){missile.missile();}

@Override

**public** **void** actionknife() {knife.knife();}

**public** **void** setFly(IFly fly) {**this**.fly = fly;}

**public** **void** setMissile(IMissile missile) {**this**.missile = missile;}

**public** **void** setKnife(IKnife knife) {**this**.knife = knife;}

}

**public** **class** LowRobot **extends** Robot{

**private** IFly fly;

**private** IMissile missile;

**private** IKnife knife;

**public** LowRobot() {

setFly(**new** FlyNo());

setMissile(**new** MissileNo());

setKnife(**new** KnifeNo());

}

@Override

**public** **void** actionFly(){fly.fly();}

@Override

**public** **void** actionMissile(){missile.missile();}

@Override

**public** **void** actionknife() {knife.knife();}

**public** **void** setFly(IFly fly) {**this**.fly = fly;}

**public** **void** setMissile(IMissile missile) {**this**.missile = missile;}

**public** **void** setKnife(IKnife knife) {**this**.knife = knife;}

}

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

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

System.***out***.println("\*\*\*\*\*SuperRobot,StandardRobot, LowRobot 주문 들어옴\*\*\*\*\*");

Robot superR = **new** SuperRobot();

Robot stanardR = **new** StandardRobot();

Robot lowR = **new** LowRobot();

Robot[] robots = {superR,stanardR,lowR};

**for**(Robot r :robots){

r.shape(); r.actionWalk(); r.actionRun();

r.actionFly(); r.actionMissile(); r.actionknife();

}//for

}//main

}//class

아래와 같이 더욱더 수정한다

**public** **abstract** **class** Robot {

**private** IFly fly;

**private** IMissile missile;

**private** IKnife knife;

**public** **void** actionWalk(){System.***out***.println("걸을 수 있습니다");}

**public** **void** actionRun(){System.***out***.println("달릴 수 있습니다");}

**public** **abstract** **void** shape();

**public** **void** actionFly(){fly.fly();}

**public** **void** actionMissile(){missile.missile();}

**public** **void** actionknife() {knife.knife();}

**public** **void** setFly(IFly fly) {**this**.fly = fly;}

**public** **void** setMissile(IMissile missile) {**this**.missile = missile;}

**public** **void** setKnife(IKnife knife) {**this**.knife = knife;}

}

/\* Super 로봇 : 날 수 O. 미사일을 쏠 수 O. 레이저검.

\* Standard로봇 : 날 수 X. 미사일을 쏠 수 O. 목검

\* Low 로봇 : 날 수 X. 미사일을 쏠 수 X. 검 없음. \*/

**public** **class** SuperRobot **extends** Robot{

**public** SuperRobot() {

setFly(**new** FlyYes());

setMissile(**new** MissileYes());

setKnife(**new** KnifeLazer());

}

@Override

**public** **void** shape() {System.***out***.println("★★★슈퍼로봇입니다. 슈퍼 팔, 다리, 몸통 머리가 있습니다");}

}

**public** **class** StandardRobot **extends** Robot{

**public** StandardRobot() {

setFly(**new** FlyNo());

setMissile(**new** MissileYes());

setKnife(**new** KnifeWood());

}

@Override

**public** **void** shape() {System.***out***.println("★★★스탠다드 로봇입니다. 스탠다드 팔, 다리, 몸통 머리가 있습니다");}

}

**public** **class** LowRobot **extends** Robot{

**public** LowRobot() {

setFly(**new** FlyNo());

setMissile(**new** MissileNo());

setKnife(**new** KnifeNo());

}

@Override

**public** **void** shape() {System.***out***.println("★★★Low 로봇입니다. Low 팔, 다리, 몸통 머리가 있습니다");}

}

/\* Super 로봇 : 날 수 O. 미사일을 쏠 수 O. 레이저검.

\* Standard로봇 : 날 수 X. 미사일을 쏠 수 O. 목검

\* Low 로봇 : 날 수 X. 미사일을 쏠 수 X. 검 없음. \*/

public class MainClass {

public static void main(String[] args) {

System.out.println("\*\*\*\*\*SuperRobot,StandardRobot, LowRobot 주문 들어옴\*\*\*\*\*");

Robot superR = new SuperRobot();

Robot stanardR = new StandardRobot();

Robot lowR = new LowRobot();

Robot[] robots = {superR,stanardR,lowR};

for(Robot r :robots){

r.shape(); r.actionWalk(); r.actionRun();

r.actionFly(); r.actionMissile(); r.actionknife();

}//for

System.out.println("\*\*\*\*\* superR의 검부품을 목검으로 교체 \*\*\*\*\*");

superR.setKnife(new KnifeWood());

superR.actionknife();

System.out.println("\*\*\*\*\* lowR의 날개 부품을 날 수 있는 것으로 교체 \*\*\*\*\*");

lowR.setFly(new FlyYes());

lowR.actionFly();

}//main

}//class

고공비행 날개 부품을 추가하고 superR의 나는 기능을 고공비행기능으로 전환하기 원한다면, 다음을 추가

**public** **class** FlyHi **implements** IFly {

@Override

**public** **void** fly() {

System.***out***.println("특별히 높이 높이 날 수 있습니다");

}

}

public class MainClass {

public static void main(String[] args) {

System.out.println("\*\*\*\*\* 고공비행 날개 부품을 추가하고 superR의 의 날개 부품을 고공비행으로 교체 \*\*\*\*\*");

superR.setFly(new FlyHi());

superR.actionFly();

}//main

}//class

<ex 2> 시나리오

모든 자동차는 기본적으로 drive할 수 있어야 합니다.

자동차 모양은 door, sheet, handle로 이루어져 있습니다.

인터페이스명 : EngineImpl(engine()), KmImpl(kmPerLitter()), FuelImpl(fuel())

Car (Genesis와 Sonata와 Casper를 상속해 줄 클래스)

engine, km, fuel, driver(), shape(), isKmPersLitter(), isFuel(), setEngine(), setKm(), setFuel()

Genesis : 최고급엔진(EngineHigh). 연비 10Km/l(Km10). 휘발유(FuelGasoline)

Sonata : 중급엔진(EngineMid). 연비 15Km/l(Km15). 휘발유(FuelGasoline)

Casper : 저급엔진(EngineLow). 연비 20Km/l(Km20). 경유(FuelDiesel).

위 3가지 형태의 차를 만들어 보도록 하고, 추후 다른 차를 만들거나 차종의 부품을 만들어야 할 경우를 생각하여 유연한 프로그램을 만들어 보도록 합니다.

**public** **interface** EngineImpl {

**public** **void** engine();

}

**public** **class** EngineHigh **implements** EngineImpl {

@Override

**public** **void** engine() {

System.***out***.println("최고급 엔진입니다");

}

}

**public** **class** EngineMid **implements** EngineImpl {

@Override

**public** **void** engine() {

System.***out***.println("중급 엔진입니다");

}

}

**public** **class** EngineLow **implements** EngineImpl {

@Override

**public** **void** engine() {

System.***out***.println("저급 엔진입니다");

}

}

**public** **interface** KmImpl {

**public** **void** kmPerLiter();

}

**public** **class** Km10 **implements** KmImpl {

@Override

**public** **void** kmPerLiter() {

System.***out***.println("연비가 10km/l입니다");

}

}

**public** **class** Km15 **implements** KmImpl {

@Override

**public** **void** kmPerLiter() {

System.***out***.println("연비가 15km/l입니다");

}

}

**public** **class** Km20 **implements** KmImpl {

@Override

**public** **void** kmPerLiter() {

System.***out***.println("연비가 20km/l입니다");

}

}

**public** **interface** FuelImpl {

**public** **void** fuel();

}

**public** **class** FuelGasoline **implements** FuelImpl {

@Override

**public** **void** fuel() {

System.***out***.println("휘발유 연료를 사용합니다");

}

}

**public** **class** FuelDiesel **implements** FuelImpl {

@Override

**public** **void** fuel() {

System.***out***.println("경유 연료를 사용합니다");

}

}

**public** **abstract** **class** Car {

**private** EngineImpl engine;

**private** KmImpl km;

**private** FuelImpl fuel;

**public** **void** drive() {

System.***out***.println("드라이브 할 수 있습니다");

}

**public** **abstract** **void** shape();

**public** **void** isEngine() { engine.engine();}

**public** **void** isKmPerLitter() {km.kmPerLiter();}

**public** **void** isFuel() {fuel.fuel();}

**public** **void** setEngine(EngineImpl engine) {**this**.engine = engine;}

**public** **void** setKm(KmImpl km) {**this**.km = km;}

**public** **void** setFuel(FuelImpl fuel) {**this**.fuel = fuel;}

}

**public** **class** Casper **extends** Car {

**public** Casper() {

// engine, km, fuel setting

setEngine(**new** EngineLow());

setKm(**new** Km20());

setFuel(**new** FuelDiesel());

}

@Override

**public** **void** shape() {

System.***out***.println("★ 소나타 차량은 문, 시트, 핸들로 이루어져 있습니다");

}

}

**public** **class** Genesis **extends** Car {

**public** Genesis() {

// engine, km, fuel setting

setEngine(**new** EngineHigh());

setKm(**new** Km10());

setFuel(**new** FuelGasoline());

}

@Override

**public** **void** shape() {

System.***out***.println("★ 제네시스 차량은 문, 시트, 핸들로 이루어져 있습니다");

}

}

**public** **class** Sonata **extends** Car {

**public** Sonata() {

// engine, km, fuel setting

setEngine(**new** EngineMid());

setKm(**new** Km15());

setFuel(**new** FuelGasoline());

}

@Override

**public** **void** shape() {

System.***out***.println("★ 소나타 차량은 문, 시트, 핸들로 이루어져 있습니다");

}

}

**public** **class** TestMain {

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

Genesis genesis = **new** Genesis();

Sonata sonata = **new** Sonata();

Casper casper = **new** Casper();

Car[] cars = {genesis, sonata, casper};

**for**(Car car : cars) {

car.shape();

car.drive();

car.isEngine();

car.isKmPerLitter();

car.isFuel();

}

}

}

main()함수내에 소나타의 연료를 전기하이브리드(FuelHybrid)로 바꾸고 연비를 20km/l(Km20)로 바꾸는 부분을 추가합니다.

**public** **class** FuelHybrid **implements** FuelImpl {

@Override

**public** **void** fuel() {

System.***out***.println("하이브리드 연료로 움직입니다");

}

}

**public** **class** TestMain {

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

Genesis genesis = **new** Genesis();

Sonata sonata = **new** Sonata();

Casper casper = **new** Casper();

Car[] cars = {genesis, sonata, casper};

**for**(Car car : cars) {

car.shape();

car.drive();

car.isEngine();

car.isKmPerLitter();

car.isFuel();

}

System.***out***.println("Sonata의 FuleHybrid로 교체하고 연비를 20으로 업그레이드");

sonata.setFuel(**new** FuelHybrid());

sonata.setKm(**new** Km20());

**for**(Car car : cars) {

car.shape();

car.drive();

car.isEngine();

car.isKmPerLitter();

car.isFuel();

}

}

}