어댑터 패턴

* 이미 구현된 클래스에 adapter 의 이름으로 상속받아 요구사항에 맞게 수정함(단 불러오는 main 클래스같은경우 변경 x) ⇒ 많은곳에서 불러올시 변경없이 알고리즘, 파라미터 등을 바꿀 수 있어야됨.
* 알고리즘을 요구사항에 맞게 변경해서 사용할 수 잇다.

ex)

package designpattern;

public class adapterPattern {

// 수의 두배 반환 수의 반의수를 반환 구현 객체이름 addapter 요구사항 float

// 메인 함수의 수정 x ==> 여러 곳에서 가저와야할시 가저오는곳은 변경 x

public static void main(String[] args) {

AdapterImpl adapter = new AdapterImpl();

System.out.println(adapter.twiceOf(100f));

System.out.println(adapter.halfOf(80f));

}

}

// 기존 클래스

class Math{

//두배 절반

public static double twoTime(double num){return num\*2; }

public static double half(double num){return num/2; }

// 강화된 알고리즘

public static Double doubled(Double d){return d\*2;} // 알고리즘이 바꼇다고 가정

}

interface Adapter{

// 원하는 기능 요청

public Float twiceOf(Float f);

public Float halfOf(Float f);

}

// 요청 해결을 위해

class AdapterImpl implements Adapter{

@Override

// public Float twiceOf(Float f) {

// return (float)Math.twoTime(f.doubleValue());

// }

// 알고리짐으 바뀐걸 가정하여 위의 수의 변경 필요

public Float twiceOf(Float f){

return Math.doubled(f.doubleValue()).floatValue();

} // 매인함수의 변경 없이 바뀐 알고리즘을 수정

@Override

public Float halfOf(Float f) {// 로그기록도 요청시

System.out.println("절반 함수 호출 시작 ");

return (float)Math.half(f.doubleValue());

}

}