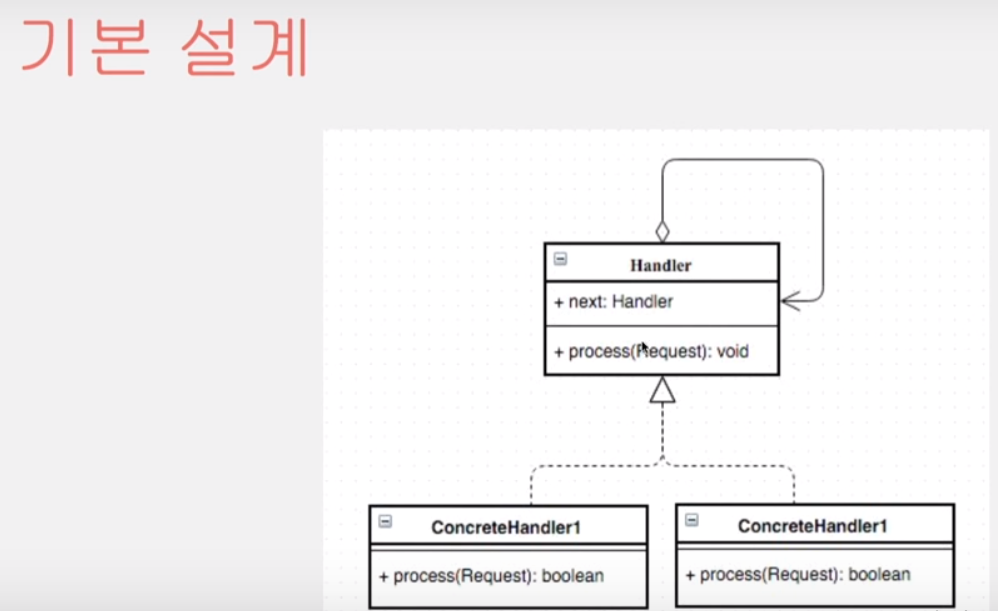
ChainofResposibility

* 다양한 처리방식을 유연하게 처리할 수 있다



# 처리를 해보고 안되면 다음으로 넘겨주는 방식으로 처리하는 소스코드

package designpattern;

public class ChainofResposibility {

public static void main(String[] args) {

Calculator plus = new PlusCalculator();

Calculator sub = new SubCalculator();

plus.setNextCalculator(sub);

Request requests1 = new Request(1,2,"+");

Request requests2 = new Request(10,2,"-");

plus.process(requests1);

plus.process(requests2);

}

}

abstract class Calculator{

private Calculator nextCalculator;

public void setNextCalculator(Calculator nextCalculator) {

this.nextCalculator = nextCalculator;

}

public Calculator getNextCalculator() {

return nextCalculator;

}

public boolean process(Request request){

if(operator(request)){

//더하기 작업

return true;

}else{

if(nextCalculator != null){

return nextCalculator.process(request);

}

return false;

}

}

//동작

abstract protected boolean operator(Request request);

}

class Request{

private int a,b;

private String oprator;

public int getA() {

return a;

}

public void setA(int a) {

this.a = a;

}

public int getB() {

return b;

}

public void setB(int b) {

this.b = b;

}

public String getOprator() {

return oprator;

}

public void setOprator(String oprator) {

this.oprator = oprator;

}

public Request(int a, int b, String oprator) {

super();

this.a = a;

this.b = b;

this.oprator = oprator;

}

}

class PlusCalculator extends Calculator{

@Override

protected boolean operator(Request request) {

if(request.getOprator().equals("+")){

int a = request.getA();

int b = request.getB();

System.out.println(a+"+"+b+"="+(a+b));

}

return false;

}

}

class SubCalculator extends Calculator{

@Override

protected boolean operator(Request request) {

if(request.getOprator().equals("-")){

int a = request.getA();

int b = request.getB();

System.out.println(a+"-"+b+" = "+ (a-b));

}

return false;

}

}

확장된 개념

package designpattern;

import java.awt.geom.Area;

public class ChainofResposibility2 {

//게임에서 공격을 주고받는 것을 책임사슬 패턴을 이용해서 이용

// 효율적이지는 않으나 이해하기위해 만듬

public static void main(String[] args) {

Attack attack = new Attack(100);

Amor amor1 = new Amor(10);

Amor amor2 = new Amor(15);

//첫번재 공격

amor1.setNextDefense(amor2);

amor1.defense(attack);

System.out.println(attack.getAmount());

//동적 point2

Defense defense = new Defense() {

@Override

public void defense(Attack attack) {

int amount = attack.getAmount();

attack.setAmount(amount-=50);

}

};

//게임 도중 추가 착용 두번째 공격

amor2.setNextDefense(defense);

attack.setAmount(100);

amor1.defense(attack);

System.out.println(attack.getAmount());

//동적으로 책임사슬을 더 추가하여 사용할 수 있음 !

}

}

class Attack{

private int amount;

public Attack(int amount){

this.amount=amount;

}

public void setAmount(int amount) {

this.amount = amount;

}

public int getAmount() {

return amount;

}

}

interface Defense{

public void defense(Attack attack);

}

class Amor implements Defense{

private Defense nextDefense;

private int def;

public Amor(int def){

this.def = def;

}

public void setNextDefense(Defense nextDefense) {

this.nextDefense = nextDefense;

}

public void setDef(int def) {

this.def = def;

}

@Override

public void defense(Attack attack) {

//중요 처리 부분

proccess(attack); //처리를 안해주고 무조건 호출

if(nextDefense!=null){

nextDefense.defense(attack);

}

}

private void proccess(Attack attack) {

int amount = attack.getAmount();

amount -= def;

attack.setAmount(amount);

}

}