Command

특정 기능들(인보커, 리시버, 커맨드)을 캡슐화 시키는 패턴->결합도. 매개변수를 이용해서 기능에 다른 요구 사항들을 넣을 수 있다.

Invoker (호출자, 기능 실행 요청), Receiver (수신자, 명령 수행 역할), Command (명령) 객체.

Command 객체들은 모두 같은 인터페이스 아래서 구현된다. Command 인터페이스에는 하나의 메소드만이 있습니다. (=execute())

리시버 객체의 동작이 늘어날 때 마다 커맨드 클래스가 늘어나기 때문에 클래스가 많아진다.

public interface Command {

    public void execute();

}

public class ComputerOnCommand implements Command{

    private Computer computer;

    public ComputerOnCommand(Computer *computer*) {

        this.computer = computer;

    }

    @Override

    public void execute() {

        computer.turnOn();

    }

}

public class ComputerOffCommand implements Command {

    private Computer computer;

    public ComputerOffCommand(Computer *computer*) {

        this.computer = computer;

    }

    @Override

    public void execute() {

        computer.turnOff();

    }

}

public class Button {

    private Command command;

    public Button(Command *command*) {

        this.command = command;

    }

    public void setCommand(Command *command*) {

        this.command = command;

    }

    public void pressButton() {

        this.command.execute();

    }

}

public static void main(String[] args) {

    Computer computer = *new* Computer(); *//컴퓨터는 리시버*

*//컴퓨터 객체 생성*

    ComputerOnCommand computerOnCmd = *new* ComputerOnCommand(computer);

    ComputerOffCommand computerOffCmd = *new* ComputerOffCommand(computer);

    Button btn = *new* Button(computerOnCmd); *//버튼이 인보커 역할*

    btn.pressButton();

    btn.setCommand(computerOffCmd);

    btn.pressButton();

}

Factory

객체의 생성을 위해서 인터페이스를 정의하지만 객체의 생성에 대해서는 서브클래스가 결정하게 만드는 패턴.

package Factory;

public interface Airplane { *//product*

    public void takeOff();

    public void fly();

    public void land();

}

package Factory;

abstract class AirplaneFactory {

    public Airplane preparingAirplane(String *planeType*) {

        Airplane airplane = createAirplane(planeType);

        airplane.takeOff();

        airplane.fly();

        airplane.land();

*return* airplane;

    }

*//생성담당*

    abstract Airplane createAirplane(String *planeType*);

}

package Factory;

public class Airliner extends AirplaneFactory {

    @Override

    Airplane createAirplane(String *planeType*) {

*return* null;

    }

}

package Factory;

public class Fighter extends AirplaneFactory {

    @Override

    Airplane createAirplane(String *planeType*) {

*return* null;

    }

}

package Factory;

public class AirPortSystem {

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

        AirplaneFactory airlinerFact = *new* Airliner();

        AirplaneFactory fighterFact = *new* Fighter();

        Airplane airplane1 = airlinerFact.createAirplane("보잉");

        Airplane airplane2 = fighterFact.createAirplane("F-15");

    }

}