

소프트웨어 프로젝트II

옵저버 패턴 구현해보기

1분반, 9조

20181582 김명환

목차

1. 프로그램 설계
2. 프로그램 코드
3. 실행 화면
4. 생각해본 점
5. 추가해보고 싶은 기능

프로그램 설계

간단한 프로그램으로 날씨 정보를 받아오는 프로그램을 생각했습니다.

처음엔 Observer 패턴과, MVC 패턴을 헷갈려서 MVC 패턴으로 프로그램을 설계했습니다.

Controller에서는 사용자 입력 값을 받아 이벤트로 Model로 넘기고, 모델에서 계산한 값을 컨트롤러에 보내고, 다시 뷰로 보내는 방식으로 진행했습니다.

프로그램을 작성하던 도중, MVC 패턴 = Observer 패턴? 이라는 의문이 들어 인터넷에 검색을 하고, 다르다는 것을 알게 되고 MVC 패턴으로 Observer 패턴을 구현하려고 했습니다.

하지만 머릿속에서 MVC 패턴과 Observer 패턴이 혼동되어 결국, Observer 패턴으로 새 프로그램을 작성했습니다.

Subject(Observing)와 Observer(Object), View, Subscriber 네 파일로 나뉘었습니다.

콘솔 창에서 출력을 할 경우, View가 콘솔이지만, Print 관련된 부분을 View 파일로 몰아 넣었습니다.

Subject 속 값만 변경되면 Observer는 아무런 변경 없이 이벤트로 호출하여 View로 출력하는 프로그램입니다.

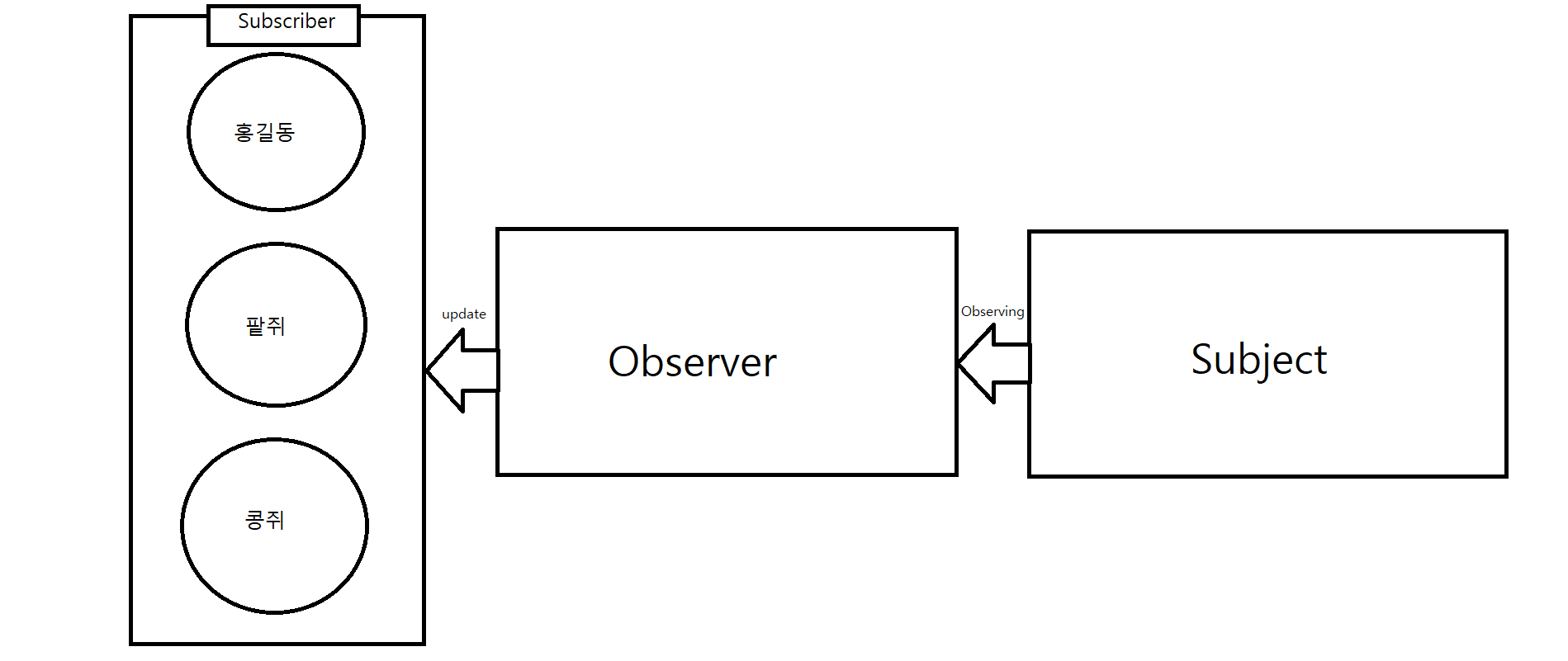
Subscriber는 구독자를 의미하며, 구독자들에게 Subject를 출력합니다.

Observer는 오직 Subject를 관찰만 하며, Subject는 Observe에게 관찰만 당합니다.

프로그램 코드

Observer\_Controller, Observer\_Model, Observer\_View는 MVC 패턴으로 작성하다가 급하게 Observer 패턴으로 돌린 파일 입니다.

완성된 프로그램에는 의미가 없을 지 몰라도, 이렇게 많이 고민했다는 것을 교수님께 알려드리고 싶습니다.

Observer, Subject, View, Subscriber가 완성된 프로그램 입니다. 

전체적인 프로그램 흐름을 그림판으로 그려봤습니다.

코드는 너무 긴 나머지, 가독성을 위해 다음 페이지 한장으로 넘긴 점 양해 부탁드립니다.

import datetime as d  
import urllib.request  
import json  
  
class Subject:  
 def \_\_init\_\_(self):  
 self.today = d.datetime.today()  
 self.ts = self.today.timetuple()  
 self.url = 'http://52.78.1.91/weather?date={}{}{}&time={}{}'.format(self.ts.tm\_year, self.ts.tm\_mon, self.ts.tm\_mday, self.ts.tm\_hour - 2, self.ts.tm\_min)  
 self.u = urllib.request.urlopen(self.url)  
 self.data = self.u.read()  
 self.j = json.loads(self.data)  
 self.temp = self.j["temperature"]  
 self.humidity = self.j["humidity"]  
 self.weather = "미구현"  
  
  
 def replaceDay(self):  
 today = d.datetime.today()  
 replaceYear = int(input("바꿀 년을 입력해주세요 : "))  
 replaceMonth = int(input("바꿀 월을 입력해주세요 : "))  
 replaceDay = int(input("바꿀 일을 입력해주세요 : "))  
 replaceHour = int(input("바꿀 시간을 입력해주세요 : "))  
 replaceMinute = int(input("바꿀 분을 입력해주세요 : "))  
 replaceSecond = int(input("바꿀 초를 입력해주세요 : "))  
 self.today = today.replace(year=replaceYear, month=replaceMonth, day=replaceDay, hour=replaceHour, minute=replaceMinute, second=replaceSecond)  
 self.ts = self.today.timetuple()  
  
  
 def replaceWeather(self):  
 self.weather = str(input("날씨를 입력해주세요 : "))  
   
   
 # 해당 부분을 json 코드로 구현해서 그런지 int로 수정하려니 에러 발생.  
 # 해당 오류 수정하려고 애썼는데 실패.  
 # 중요한건 옵저버 패턴이니 옵저버 패턴을 구현하려고 시도한 점에 의의를 가지자.  
 def replaceTempHumidity(self):  
 self.replaceTemp = int(input("바꿀 온도를 입력해주세요 : "))  
 self.replaceHumidity = int(input("바꿀 습도를 입력해주세요 : "))  
  
 def todayNow(self):  
 self.today = d.datetime.today()  
 self.ts = self.today.timetuple()  
  
 def weatherNow(self):  
 self.weather = "미구현"  
  
 def tempHumidityNow(self):  
 self.url = 'http://52.78.1.91/weather?date={}{}{}&time={}{}'.format(self.ts.tm\_year, self.ts.tm\_mon, self.ts.tm\_mday, self.ts.tm\_hour - 2, self.ts.tm\_min)  
 self.u = urllib.request.urlopen(self.url)  
 self.data = self.u.read()  
 self.j = json.loads(self.data)  
 self.temp = self.j["temperature"]  
 self.humidity = self.j["humidity"]

Subject.py 코드이다.

날씨 프로그램을 작성하던 도중, ‘날짜(시간)마다 날씨가 바뀌니까 인터넷 url을 json코드로 받아서 출력시키는 것이 어떨까?’ 생각을 했습니다.

인터넷에 json을 받아오는 방법을 검색해 1학기 소프트웨어 프로젝트 I 때 쓰였던 url을 이용해 작성했습니다.

해당 url은 {"code": "success", "temperature": 13.2, "humidity": 31} 값을 가지고 있으며, json을 이용해 필요한 정보인 temperature, humidity만 가지고 왔습니다.

하지만 해당 url은 정보 저장 기간이 짧아, 내가 날짜를 2001년 12월 1일로 바꾼다면 분명 오류가 발생할 것 입니다.

해당 부분까지 오류를 해결 하기 위해선 url 항상 최신의 정보 값을 주는 api url을 받아와야 하지만, 시간이 너무 부족해서 구현하지 못했습니다.

또한 온도와 습도 뿐만 아니라 날씨 정보 까지 구현하고 싶었지만, 제가 가지고 있던 api url은 날씨 정보가 없어 구현하지 못했습니다.

짧은 시간에 배우지 못한 모듈을 import해서 사용하느라 프로그램 완성도가 많이 떨어집니다. 하지만 Observer 패턴을 공부하는 이의에서 작성한 프로그램이기 때문에 완성도가 떨어져도 교수님의 양해 부탁드립니다.

from Subject import Subject  
import View as view  
import Subscriber as subscriber  
  
def Main():  
 re = 0  
 subject = Subject()  
 while(re != 9):  
 print("==================관리자 화면=====================")  
 view.day((subject.ts.tm\_year), (subject.ts.tm\_mon), (subject.ts.tm\_mday), (subject.ts.tm\_hour), (subject.ts.tm\_min), (subject.ts.tm\_sec))  
 view.tempHumidity((subject.temp), (subject.humidity))  
 view.weather(subject.weather)  
 view.menu() # self 사용이 아니라서 model 말고 Model 사용  
 re = int(input("숫자를 입력해주세요. : "))  
  
 if re == 1:  
 subject.replaceDay()  
  
 if re == 2:  
 subject.replaceWeather()  
  
 if re == 3:  
 subject.replaceTempHumidity()  
  
 if re == 4:  
 subject.todayNow()  
  
 if re == 5:  
 subject.weatherNow()  
  
 if re == 6:  
 subject.tempHumidityNow()  
  
 subscriber.subData((subject.ts.tm\_year), (subject.ts.tm\_mon), (subject.ts.tm\_mday), (subject.ts.tm\_hour), (subject.ts.tm\_min), (subject.ts.tm\_sec),(subject.temp), (subject.humidity), (subject.weather))  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 Main()

Observer.py 코드이다.

Observer.py 코드는 매우 직관적입니다.

View.py 함수를 호출해 콘솔창을 띄우고, 사용자의 입력(이벤트)를 받으면 로직에 따라 관찰하고 있는 Subject로 넘어가 계산 값만 받고, 다시 while문을 돌면서 View로 바뀐 값을 전달합니다.

오직 이벤트를 Subject로 넘기고, 변경된 값을 받은 후, View로 넘기는 코드입니다.

관리자의 입력 값을 받으면 (이벤트가 실행되면) Subject 값이 변경되고, 그 변경 값을 Subscriber로 넘겨 모든 구독자들에게 같은 화면을 보여줍니다.

def day(year, mon, mday, hour, min, sec):  
 print("현재 시간 : {}년 {}월 {}일 {}시 {}분 {}초".format(year, mon, mday, hour, min, sec))  
  
  
def tempHumidity(temp, humidity):  
 print("현재 온도 : {}, 현재 습도 : {}".format(temp, humidity))  
  
  
def weather(weather):  
 print("날씨 : {}".format(weather))  
  
def menu():  
 print("==================관리자 메뉴=====================")  
 print("1. 날짜 수정")  
 print("2. 날씨 수정")  
 print("3. 온도 수정")  
 print("4. 날짜 갱신")  
 print("5. 날씨 갱신")  
 print("6. 온도 갱신")  
 print("9. 프로그램 종료")  
 print("================================================")

View 프로그램 전체 코드이다.

Observer, Subject 어떠한 것도 import 받지 않고 오직 출력에만 집중합니다.

def menu 같은 경우는 오직 관리자만 접근할 수 있기에 “관리자 메뉴”라고 따로 지칭했습니다.

Subscriber = ["홍길동", "콩쥐", "팥쥐"]  
  
def subData(year, mon, mday, hour, minute, sec, temp, humidity, weather):  
 for i in range (len(Subscriber)):  
 print("==============={}님의 화면입니다.================".format(Subscriber[i]))  
 print("현재 시간 : {}년 {}월 {}일 {}시 {}분 {}초".format(year, mon, mday, hour, minute, sec))  
 print("현재 온도 : {}, 현재 습도 : {}".format(temp, humidity))  
 print("날씨 : {}".format(weather))  
 print("================================================")

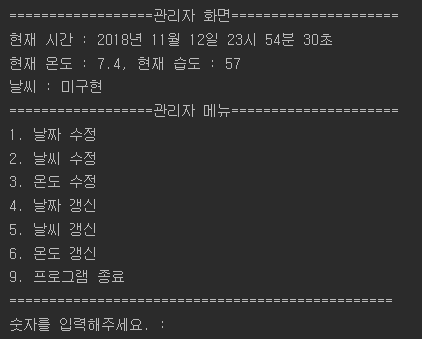
Subscriber.py 코드이다.

구독자 리스트에 존재하는 구독자들에게 같은 화면을 출력합니다.

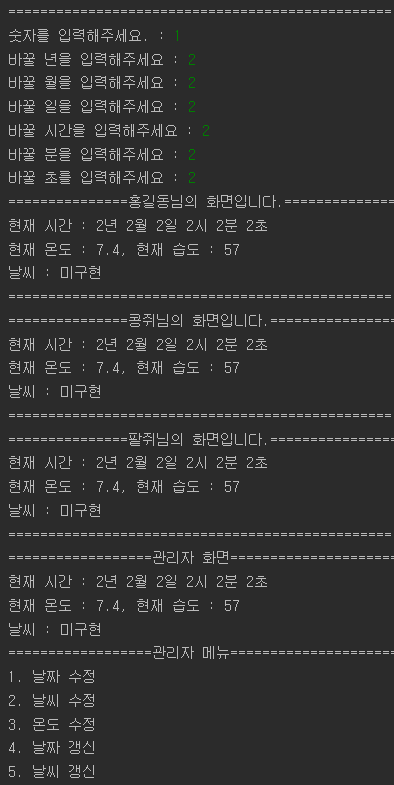
Observer.py에서 인자 값을 보내, Subscriber에서 출력하는 코드입니다.

관리자의 입력 값(event)로 인해 Subject 정보가 변경되면, 모든 구독자들에게 변경된 값을 보내, 화면을 출력합니다.

실행 화면

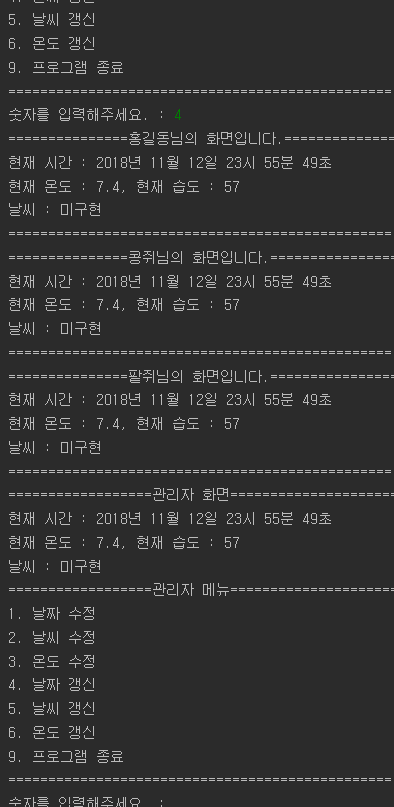


초기 실행 화면이다.



날짜를 2년 2월 2일 2시 2분 2초로 바꾼 화면이다.

관리자가 Subject의 값을 변경한 화면이다. 구독자들에겐 동일한 화면이 출력된다.



현재 시간으로 날짜를 갱신한 화면이다.

관리자가 현재 시간으로 날짜를 갱신하자, 모든 구독자들의 화면에도 날짜가 갱신되었다.

생각해본 점

Observer 패턴은 구독자가 다수일 경우, Subject 값만 변경하면 모든 구독자에게 같은 화면이 출력되기에 특정 상황에서 매우 간편하다!

객체의 상태가 바뀔 때 다른 객체들에게 그 사실을 알려주고, 자동적으로 업데이트 되도록 하고자 할 때, 이 패턴을 사용하면 매우 유용할 것이다.

Observer 패턴은 MVC 패턴에서 Model과 View 사이가 느슨해진 것이다.

다수의 구독자들에게 프로그램 화면을 출력해주는 경우엔 Observer 패턴을 사용하자!

MVC 패턴은 다수의 구독자를 상대하기엔 버거운 것 같다. (Model 하위 계층이 View 상위 계층에 종속될 가능성이 높다.)

Observer 패턴에서 구독자들에게 값을 update 해주는 방식도 2가지가 있다고 한다.

Pull update -> Subject가 편함!

Why? -> 변경 사실만 알려주면 되니까!

Push update -> Observer가 편함!

하지만 Push update를 무턱대고 한다면 Observer가 편하지 않음. (Push update가 어떤 방식인지 잘 모르겠다.)

내가 작성한 프로그램은 Pull update로 추정된다.

또 다른 디자인 패턴은 없을까? -> 생성 패턴, 구조 패턴, 행위 패턴 등.. 매우 많다.

MVC 패턴에서 데이터가 변하면 뷰가 자동으로 갱신되는 것이 Observer 패턴을 활용한 것!

감사합니다.