모델 뷰 컨트롤러 패턴

이관우

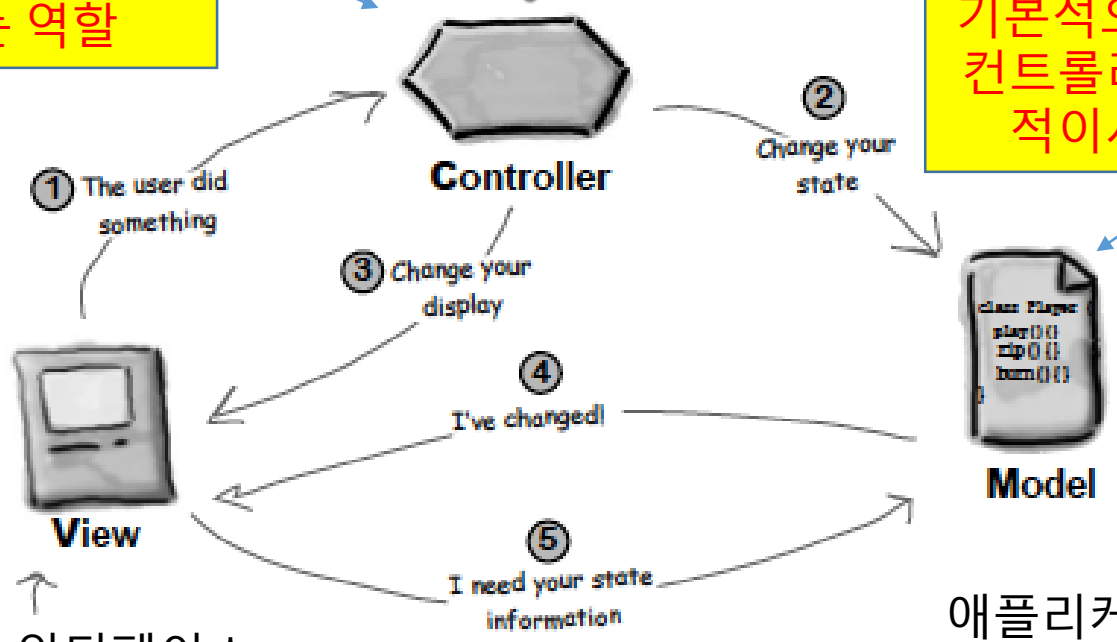
kwlee@hansung.ac.kr

MVC 개요

제어로직을 뷰로부터 분리,
뷰와 모델의 결합을 끊어주는 역할

사용자로부터 입력을 받고, 그 의미를 파악하여 모델의 조작을 결정

기본적으로, 모델은 뷰나 컨트롤러에 대해서 의존적이서는 안됩니다.

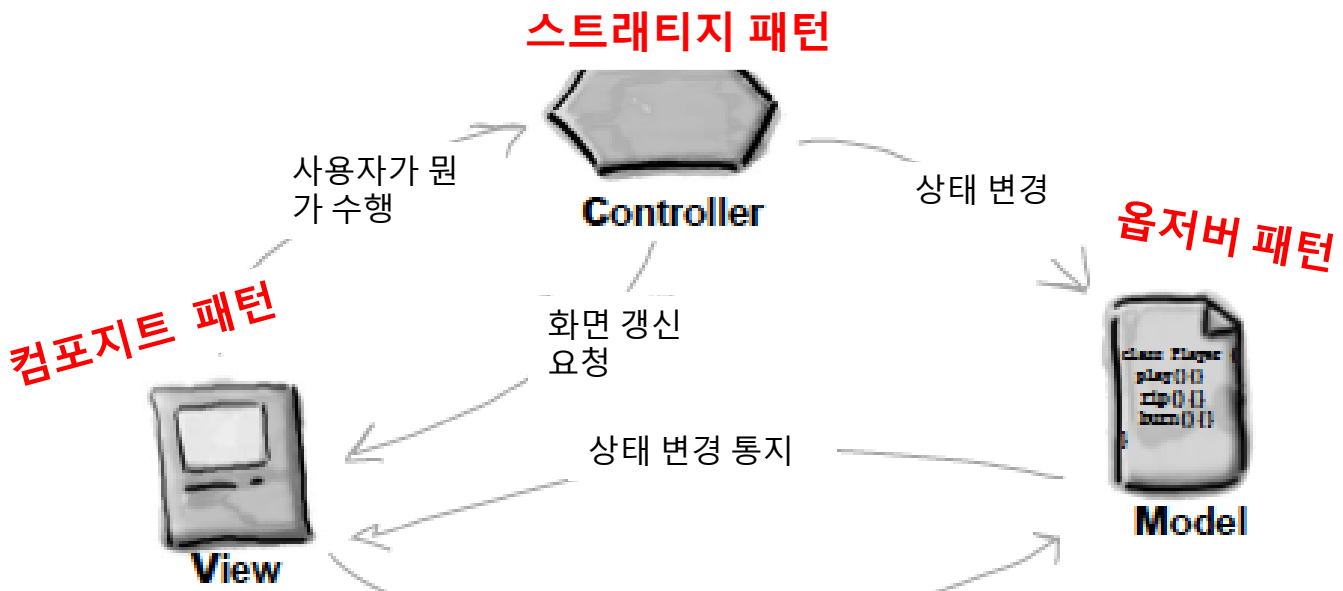


사용자 인터페이스

애플리케이션 데이터 및 로직을 처리

MVC와 패턴의 관계

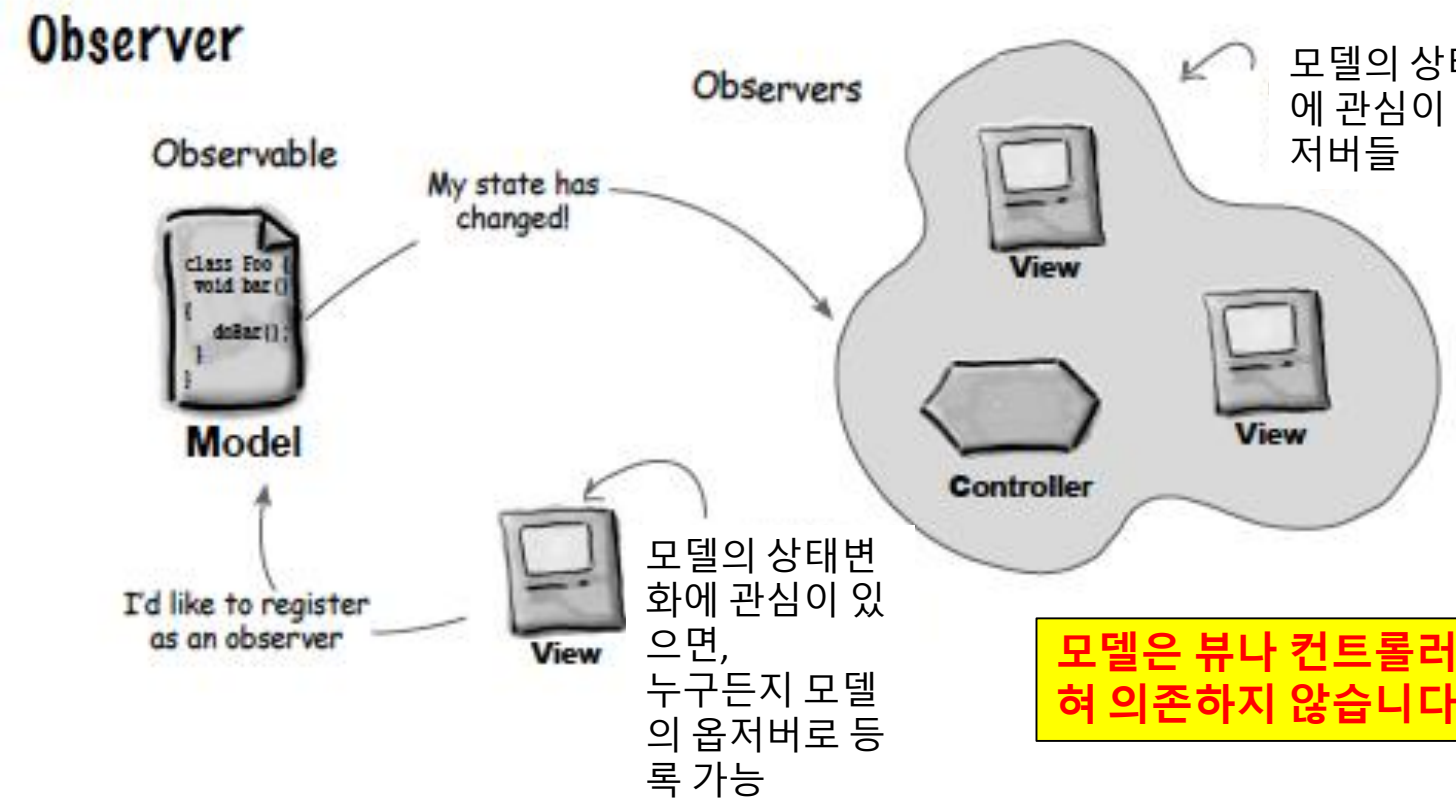
뷰에서는 애플리케이션의 겉모습만 신경을 쓰고, 인터페이스의 행동에 대한 결정은 모두 컨트롤러한테 맡김
스트래티지 패턴은 모델을 뷰로부터 분리시키는데 도움이 됨



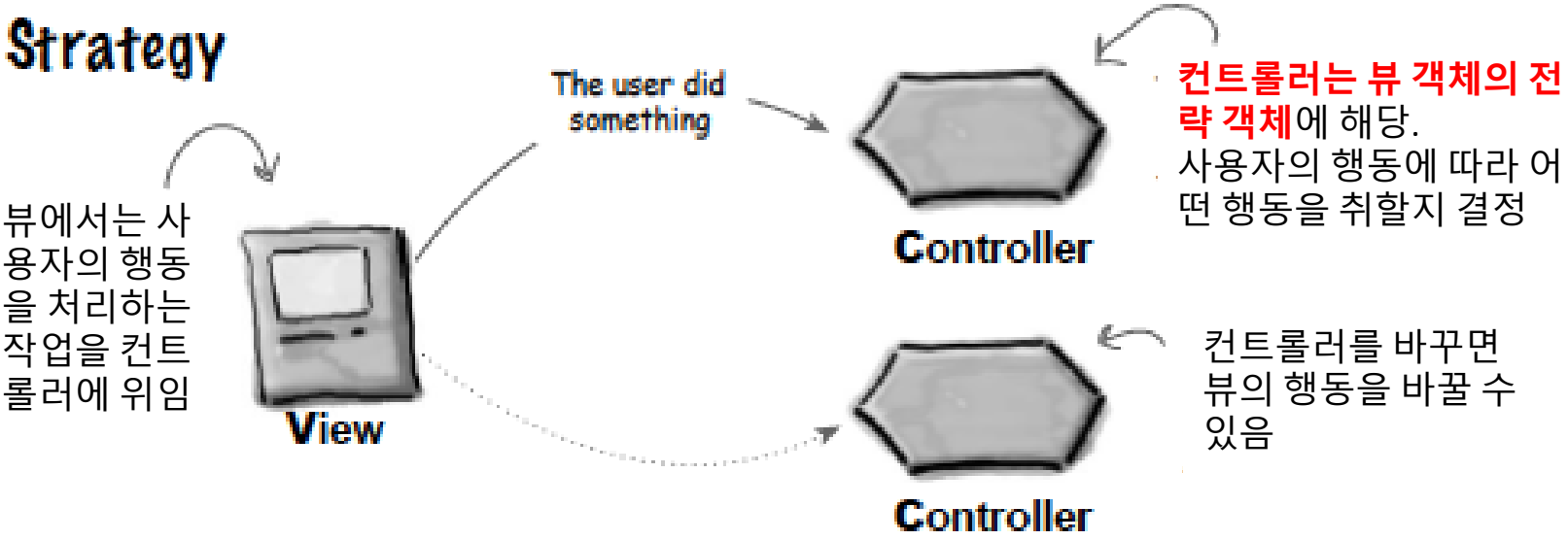
각 디스플레이 항목은 복합 객체 또는 개별 객체로 구성
컴포지트 패턴은 디스플레이 항목을 동일한 방식으로 사용가능하도록 함

모델의 상태가 변경되었을 때, 연관된 객체에게 통지
옵저버 패턴은 모델을 뷰 및 컨트롤러로부터 완전히 독립시킬 수 있음

MVC와 옵저버 패턴


Observer
Observable
My state has changed!
Observers
모델의 상태 변경에 관심이 있는 옵저버들
class Foo {
void bar()
{
doBar();
}
}
Model
View
View
Controller
I'd like to register as an observer
View
모델의 상태변화에 관심이 있으면, 누구든지 모델의 옵저버로 등록 가능
모델은 뷰나 컨트롤러한테 전혀 의존하지 않습니다.

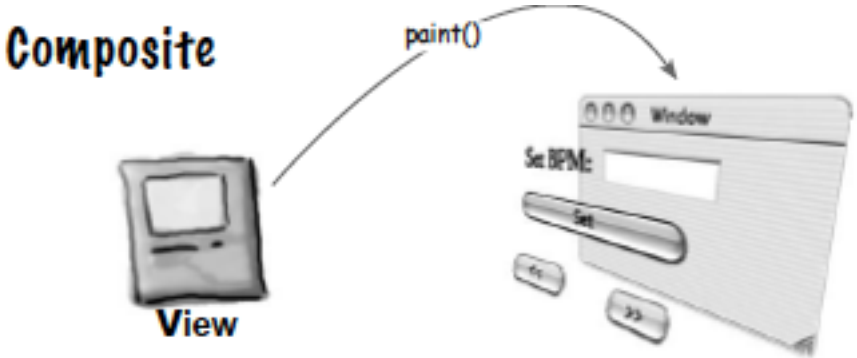
MVC와 스트래티지 패턴



MVC와 컴포지트 패턴

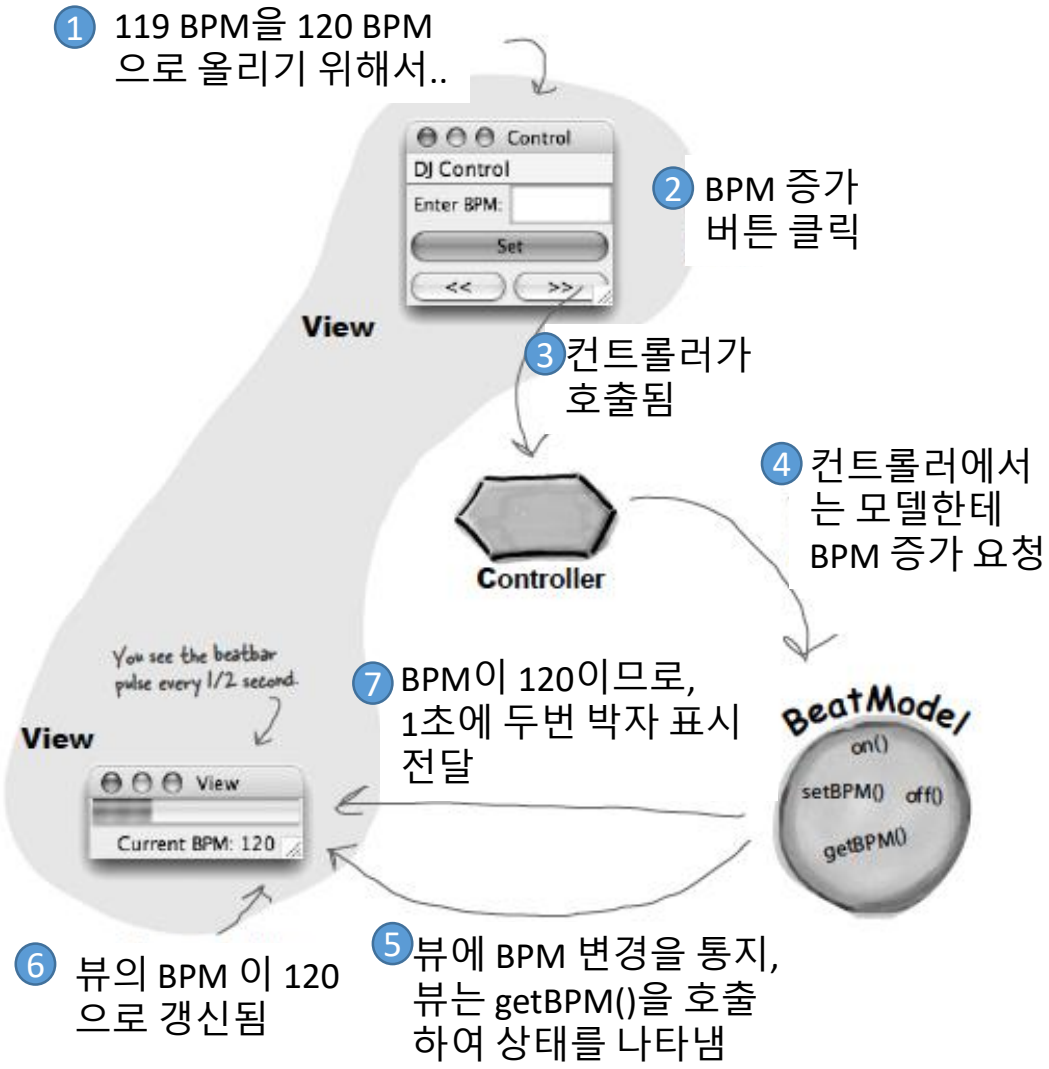
뷰는 GUI 구성요소 (레이블, 버튼, 텍스트 항목)들로 구성된 복합 객체

최상위 구성요소에는 다른 구성요소들이 들어 있고, 그 안에는 또 다시 각각 다른 구성요소들이 들어가 있음

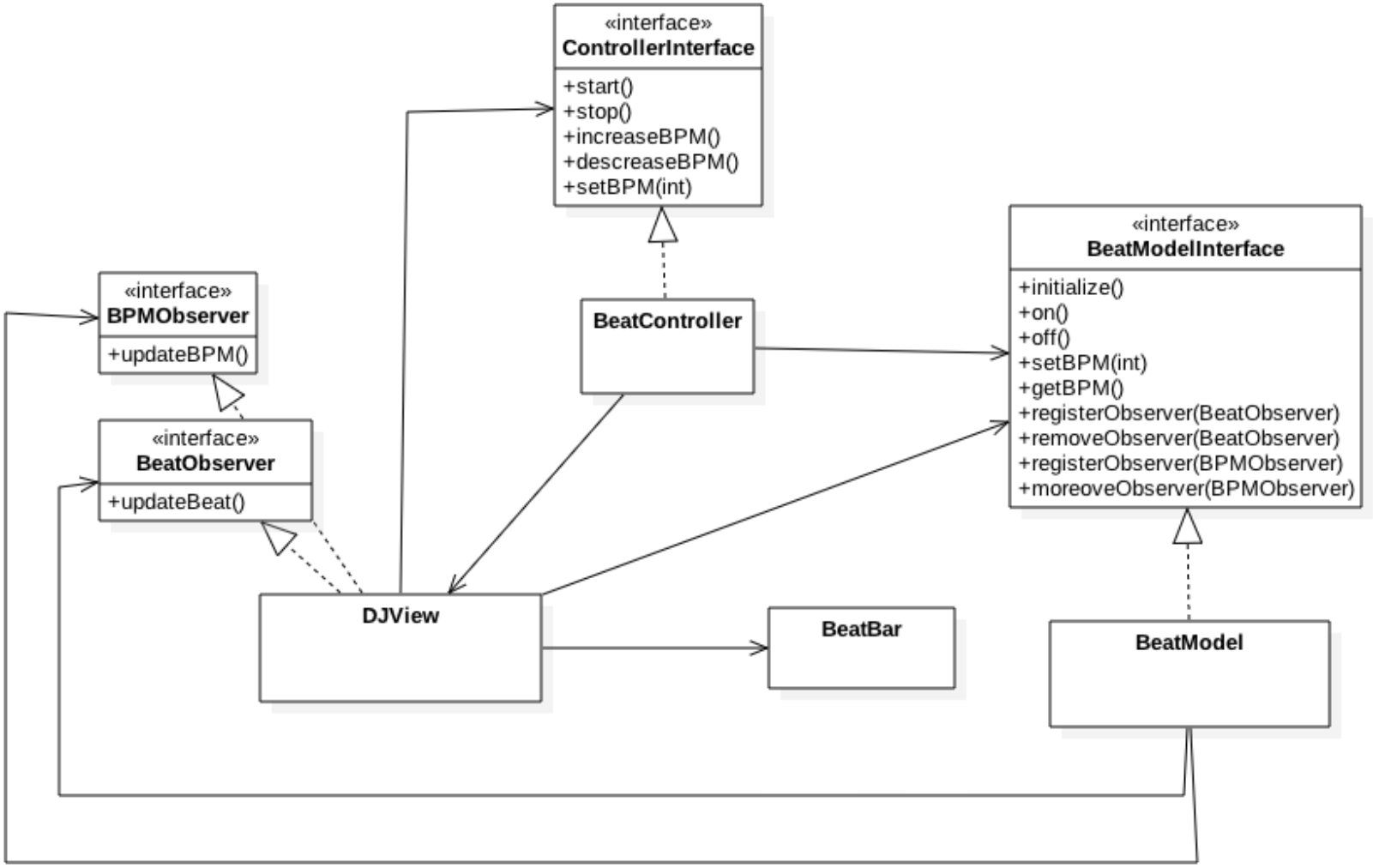


MVC를 이용한 박자 조절

https://github.com/kwanulee/DesignPattern/tree/master/mvc



전체 클래스 구조


«interface»
ControllerInterface
+start()
+stop()
+increaseBPM()
+descreaseBPM()
+setBPM(int)
«interface»
BeatModelInterface
+initialize()
+on()
+off()
+setBPM(int)
+getBPM()
+registerObserver(BeatObserver)
+removeObserver(BeatObserver)
+registerObserver(BPMObserver)
+moreoveObserver(BPMObserver)
«interface»
BPMObserver
+updateBPM()
BeatController
«interface»
BeatObserver
+updateBeat()
DJView
BeatBar
BeatModel

```
public class BeatModel implements BeatModelInterface, MetaEventListener {
    Sequencer sequencer;
    ArrayList<BeatObserver> beatObservers = new ArrayList<BeatObserver>();
    ArrayList<BPMObserver> bpmObservers = new ArrayList<BPMObserver>();
    int bpm = 90;
    // 기타 인스턴스 변수

    public void initialize() { … }
    public void on() { sequencer.start(); setBPM(90); }
    public void off() { setBPM(0); sequencer.stop(); }
    public void setBPM(int bpm) {
         this.bpm = bpm;
         sequencer.setTempoInBPM(getBPM());
         notifyBPMObservers();
    }
    public int getBPM() { return bpm;}

    void beatEvent() { notifyBeatObservers(); }
    public void registerObserver(BeatObserver o) { … }
    public void notifyBeatObservers() { … }
    public void registerObserver(BPMObserver o) { … }
    public void notifyBPMObservers() { … }
    public void meta(MetaMessage message) {
         … beatEvent(); …
     }
```

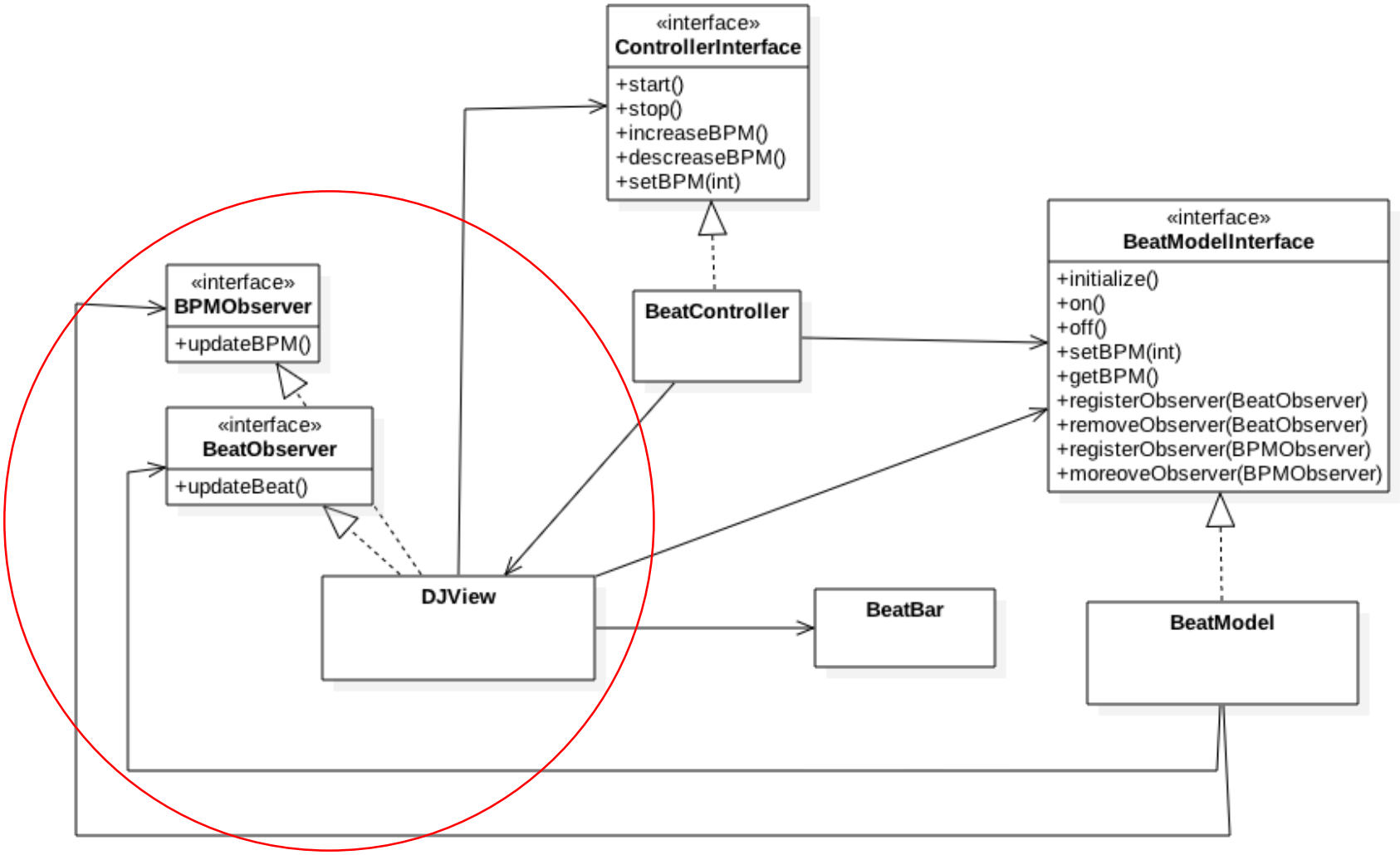
[상태변화] Controller에서 bpm값을 변경시키기 위해서, setBPM()메소드를 호출

meta() 메소드에서 이 메소드를 호출

[상태변화] 비트가 새로이 시작될 때마다 meta() 메소드가 호출됨

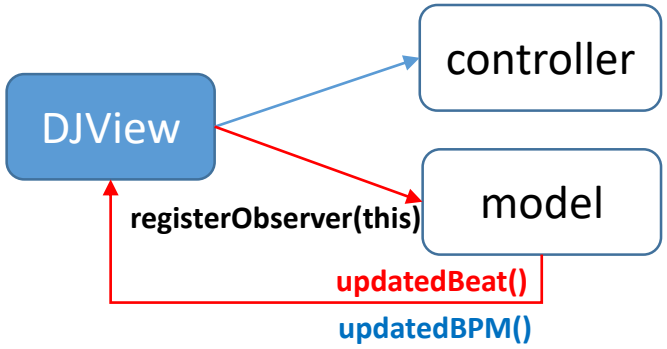
https://github.com/kwanulee/DesignPattern/blob/master/mvc/src/djview/BeatModel.java

뷰 구현하기


«interface»
ControllerInterface
+start()
+stop()
+increaseBPM()
+descreaseBPM()
+setBPM(int)
«interface»
BeatModelInterface
+initialize()
+on()
+off()
+setBPM(int)
+getBPM()
+registerObserver(BeatObserver)
+removeObserver(BeatObserver)
+registerObserver(BPMObserver)
+moreoveObserver(BPMObserver)
«interface»
BPMObserver
+updateBPM()
BeatController
«interface»
BeatObserver
+updateBeat()
DJView
BeatBar
BeatModel

뷰 구현하기 (BeatObserver, BPMOBserver 인터페이스 구현)

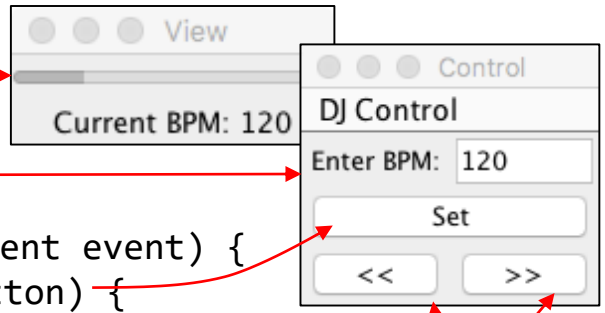
```
public class DJView implements ActionListener,  BeatObserver, BPMObserver {
    BeatModelInterface model;
    ControllerInterface controller;

    public DJView(ControllerInterface controller, BeatModelInterface model) {
      this.controller = controller;
      this.model = model;
      model.registerObserver((BeatObserver)this);
      model.registerObserver((BPMObserver)this);
     }

    public void updateBPM() {
      bpm = model.getBPM();
      if (bpm == 0)
         bpmOutputLabel.setText("offline");
      else
         bpmOutputLabel.setText("Current BPM: " + model.getBPM());
    }

    public void updateBeat() {
          beatBar.setValue(100);
    }
```



https://github.com/kwanulee/DesignPattern/blob/master/mvc/src/djview/DJView.java

뷰 구현하기 (ActionListener 인터페이스 구현)

```
public class DJView implements ActionListener,  BeatObserver, BPMObserver {

    public void createView() { … }

    public void createControls() { … }

    public void actionPerformed(ActionEvent event) {
      if (event.getSource() == setBPMButton) {
         int bpm = Integer.parseInt(bpmTextField.getText());
         controller.setBPM(bpm);
      } else if (event.getSource() == increaseBPMButton) {
         controller.increaseBPM();
      } else if (event.getSource() == decreaseBPMButton) {
         controller.decreaseBPM();
      }
    }
```



컨트롤러에서 이 메소드들을 사용하여 메뉴 항목의 활성화/비활성화 제어

```
JMenuItem startMenuItem;
JMenuItem stopMenuItem;
public void enableStopMenuItem() { stopMenuItem.setEnabled(true); }
public void disableStopMenuItem() { stopMenuItem.setEnabled(false); }
public void enableStartMenuItem() { startMenuItem.setEnabled(true); }
public void disableStartMenuItem() { startMenuItem.setEnabled(false); }
```

컨트롤러 구현

```
public class BeatController implements ControllerInterface {
        BeatModelInterface model;
        DJView view;

        public BeatController(BeatModelInterface model) {
                this.model = model;
                view = new DJView(this, model);
                view.createView();
                view.createControls();
                view.disableStopMenuItem();
                view.enableStartMenuItem();

                model.initialize();
        }

        public void start() {
                model.on();
                view.disableStartMenuItem();
                view.enableStopMenuItem();
        }

        …
```

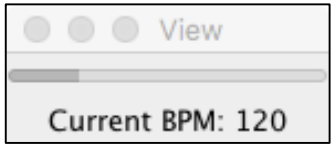
https://github.com/kwanulee/DesignPattern/blob/master/mvc/src/djview/BeatController.java

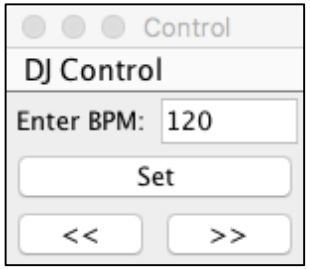
컨트롤러 구현

```
public class BeatController implements ControllerInterface {
        …

        public void stop() {
                model.off();
                view.disableStopMenuItem();
                view.enableStartMenuItem();
        }

        public void increaseBPM() {
                int bpm = model.getBPM();
                model.setBPM(bpm + 1);
        }

        public void decreaseBPM() {
                int bpm = model.getBPM();
                model.setBPM(bpm - 1);
        }

        public void setBPM(int bpm) {
                model.setBPM(bpm);
        }
}
```

테스트

```
public class DJTestDrive {

    public static void main (String[] args) {
        BeatModelInterface model = new BeatModel();
                        ControllerInterface controller = new BeatController(model);
    }
}
```

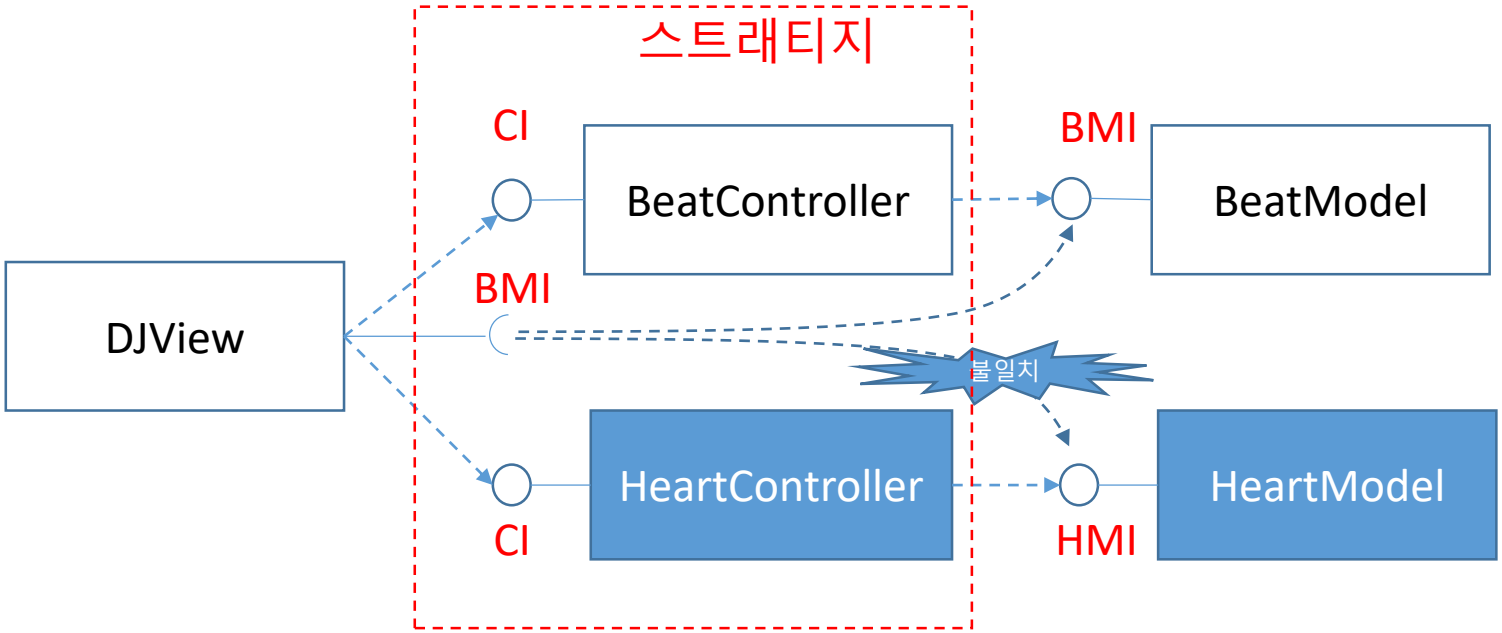




https://github.com/kwanulee/DesignPattern/blob/master/mvc/src/djview/DJTestDrive.java

스트래티지 패턴 탐색

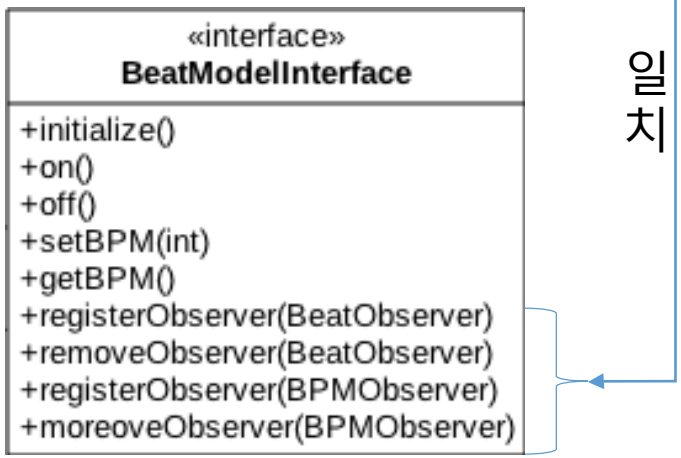
- 심장 박동 모니터링에 DJView 재사용



CI:ControllerInterface, BMI:BeatModelInterface, HMI:HeartModelInterface

HeartModelInterface

```
public interface HeartModelInterface {
        int getHeartRate();
        void registerObserver(BeatObserver o);
        void removeObserver(BeatObserver o);
        void registerObserver(BPMObserver o);
        void removeObserver(BPMObserver o);
}
```



HeartModel

```
public class HeartModel implements HeartModelInterface, Runnable {
        ArrayList<BeatObserver> beatObservers = new ArrayList<BeatObserver>();
        ArrayList<BPMObserver> bpmObservers = new ArrayList<BPMObserver>();
        …

        public void registerObserver(BeatObserver o) { beatObservers.add(o); }

        public void removeObserver(BeatObserver o) {…}

        public void notifyBeatObservers() {
            for(int i = 0; i < beatObservers.size(); i++) {
                BeatObserver observer = (BeatObserver)beatObservers.get(i);
                System.out.println("HeartModel::notifyBeatObservers()");
                observer.updateBeat();
            }
        }

        public void registerObserver(BPMObserver o) { bpmObservers.add(o); }

        public void removeObserver(BPMObserver o) { … }

        public void notifyBPMObservers() { … }
}
```

https://github.com/kwanulee/DesignPattern/blob/master/mvc/src/djview/HeartModel.java

HeartModel

```
public class HeartModel implements HeartModelInterface, Runnable {
    int time = 1000;
    Random random = new Random(System.currentTimeMillis());
    Thread thread;

    public HeartModel() {
        thread = new Thread(this); thread.start();
    }

    public void run() {
        int lastrate = -1;
        for(;;) {
            int rate = 60000/(time + change);    // change = -9~9 사이에서 변함
            notifyBeatObservers();
            …
            if (rate != lastrate) {
                lastrate=rate;
                notifyBPMObservers();
            }
            Thread.sleep(time);
        }
    }

    public int getHeartRate() { return 60000/time; }
```

매 time 밀리초 마다 호출

rate 값이 이전 값과 다를 경우에 호출

현재 rate값 반환

HeartAdapter

```
public class HeartAdapter implements BeatModelInterface {
    HeartModelInterface heart;

    public HeartAdapter(HeartModelInterface heart) {
       this.heart = heart;
    }

    public void initialize() {}
    public void on() {}
    public void off() {}
    public void setBPM(int bpm) {}

    public int getBPM() { return heart.getHeartRate(); }

    public void registerObserver(BeatObserver o){ heart.registerObserver(o); }

    public void removeObserver(BeatObserver o) { heart.removeObserver(o); }

    public void registerObserver(BPMObserver o) { heart.registerObserver(o); }

    public void removeObserver(BPMObserver o) { heart.removeObserver(o); }
}
```

https://github.com/kwanulee/DesignPattern/blob/master/mvc/src/djview/HeartAdapter.java

HeartController

```
public class HeartController implements ControllerInterface {
        HeartModelInterface model;
        DJView view;

        public HeartController(HeartModelInterface model) {
                this.model = model;

                view = new DJView(this, new HeartAdapter(model) );
                view.createView();
                view.createControls();
                view.disableStopMenuItem();
                view.disableStartMenuItem();
        }

        public void start() {}

        public void stop() {}

        public void increaseBPM() {}

        public void decreaseBPM() {}

        public void setBPM(int bpm) {}
}
```

메뉴 항목 비활성화

심장은 우리 맘대로 제어할 수 없으므로, 그대로 둠

https://github.com/kwanulee/DesignPattern/blob/master/mvc/src/djview/HeartController.java

테스트

```
public class HeartTestDrive {

    public static void main (String[] args) {
            HeartModel heartModel = new HeartModel();
            ControllerInterface model = new HeartController(heartModel);
    }
}
```

https://github.com/kwanulee/DesignPattern/blob/master/mvc/src/djview/HeartTestDrive.java

핵심 정리

- 모델 뷰 컨트롤러 패턴 (MVC)은 옵저버 패턴, 스트래티지 패턴, 컴포지트 패턴으로 이루어진 컴파운드 패턴
- 모델에서는 옵저버 패턴을 이용하여 옵저버들에 대한 의존성은 없애면서도 옵저버들한테 자신의 상태가 변경되었음을 알릴 수 있음
- 컨트롤러는 뷰에 대한 전략 객체임.
  - 뷰에서는 컨트롤러를 바꾸기만 하면 다른 행동을 활용할 수 있음
- 뷰에서는 컴포지트 패턴을 이용하여 사용자 인터페이스를 구현
- 새로운 모델을 기존의 뷰 및 컨트롤러하고 연결해서 쓸때는 어댑터 패턴을 활용