고급객체지향 프로그래밍
강의노트 #12

MVC Pattern

조용주
ycho@smu.ac.kr

MVC 패턴 (Model-View-Controller Pattern)

- 목적
  - 업무 로직 (Business Logic) 및 데이터 (Model) 와 보이는 부분 (View) 을 분리
  - 화면을 포함하는 사용자 인터페이스와 업무 처리 부분을 분리해서 동작할 수 있도록 함
- MVC 는 원래 Smalltalk 언어에서부터 시작되었으나 현재 GUI 를 지원하는 경우와 웹 프레임워크에서 많이 사용됨

디자인 패턴 요소

요소	설명
이름	MVC 혹은 Model-View-Controller
문제	데이터와 해당 데이터를 보여주는 부분의 코드가 섞여 있음
해결방안	데이터와 뷰를 분리하고 이들을 연동시키는 컨트롤러를 추가
결과	loose coupling, 재사용성

설계

역할	설계
모델 (Model)	응용 프로그램의 데이터를 관리하는 부분 또는 로직 (뷰의 모델은 한 개만 있음)
뷰 (View)	사용자가 보는 화면에 나타나는 부분을 관리 (모델에 대해서 뷰는 여러 개 있을 수 있음)
컨트롤러 (Controller)	사용자의 입력을 처리하고 , Model 과 View 사이에서 상호 작용을 할 수 있도록 지원 (한 개 이상 있을 수 있음)

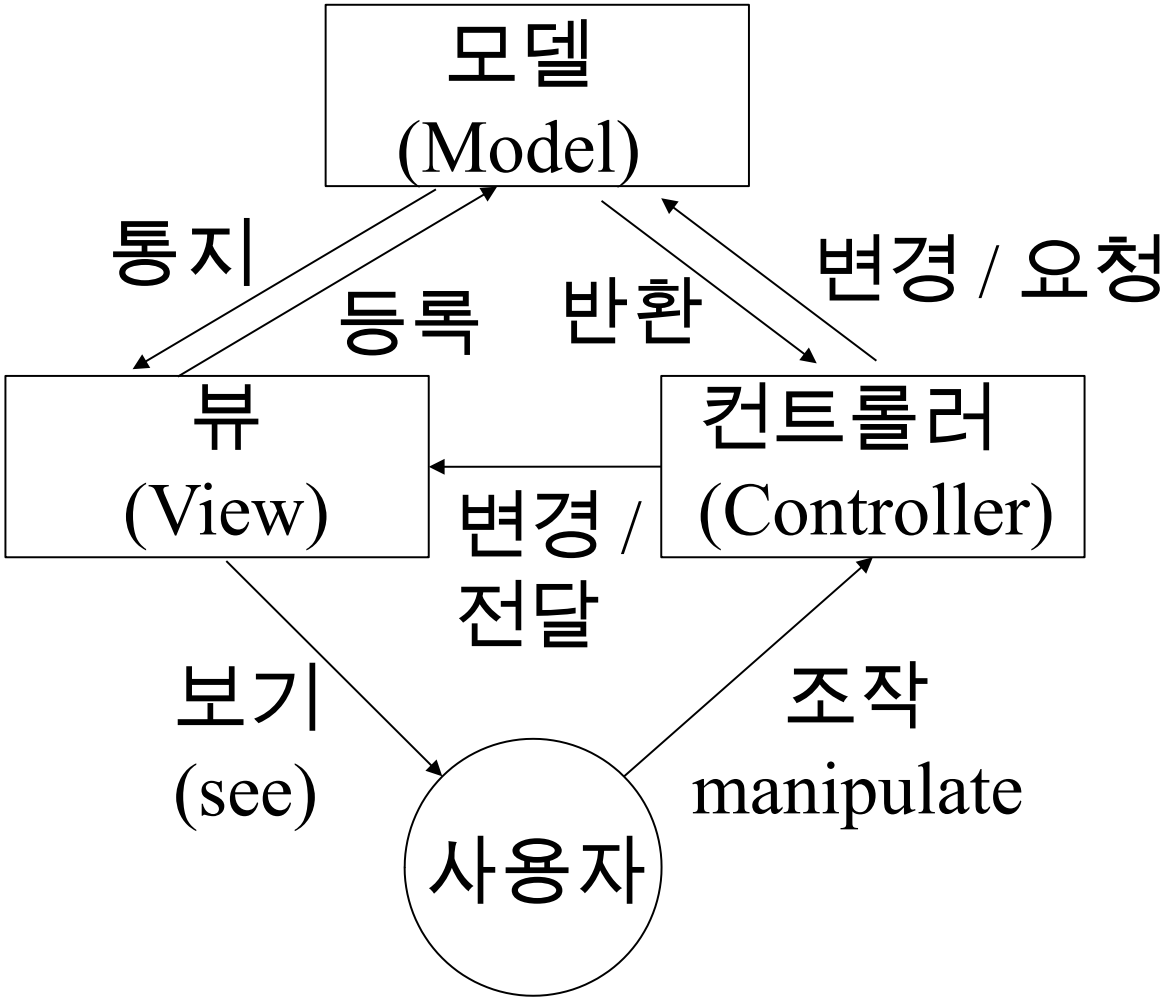
MVC 패턴의 변형

- 마이크로소프트사의 MFC
  - Document/View 구조를 사용
  - Document 는 Model 을 나타내고 , Controller 는 윈도우의 메시징 시스템에서 처리
- 자바 Swing
  - Model/Delegate 구조를 사용
  - Delegate 는 컨트롤러 + 뷰라고 생각하면 됨
  - 자바 스윙의 컴포넌트들은 모델과 델리게이트로 구성됨
    - 각 컴포넌트들은 기본적인 기능을 제공하는 디폴트 모델과 델리게이트를 제공
    - setModel() 과 setUI() 메소드를 이용해서 모델 또는 델리게이트를 변경 가능

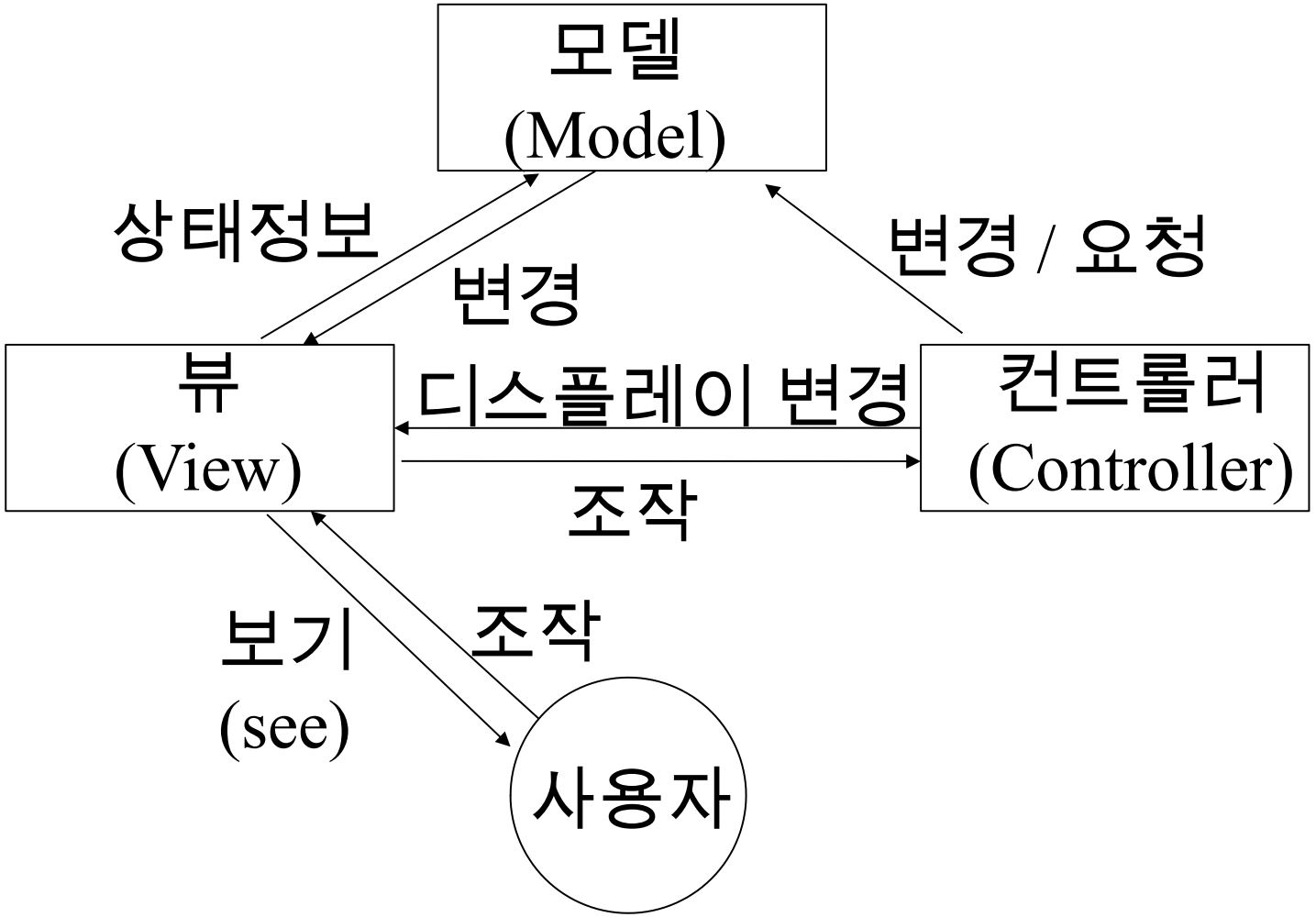
MVC 와 웹

- 생활 코딩에서 인용 (https://opentutorials.org/course/697/3828)
- 사용자가 웹사이트에 접속한다
- 컨트롤러는 사용자가 요청한 웹페이지를 서비스하기 위해서 모델을 호출
- 모델은 데이터베이스나 파일과 같은 데이터 소스를 제어한 후에 그 결과를 반환
- 컨트롤러는 모델이 반환한 결과를 뷰에 반영
- 데이터가 반영된 뷰는 사용자에게 보여짐

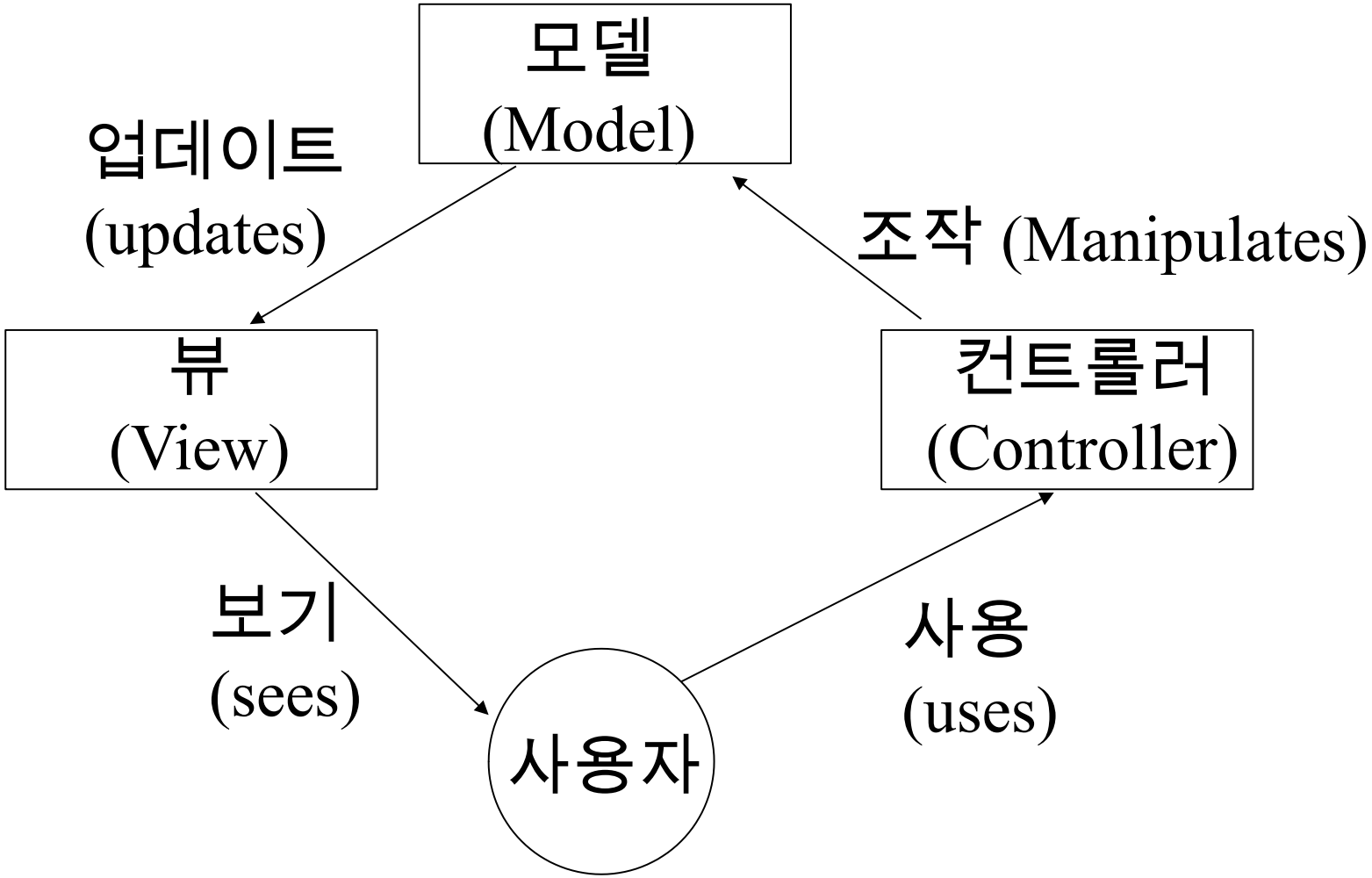
MVC 패턴 (Swing)


모델
(Model)
통지
등록
반환
변경 / 요청
뷰
(View)
컨트롤러
(Controller)
변경 /
전달
보기
(see)
조작
manipulate
사용자

MVC 패턴 – Head First



MVC 패턴 -- 생활코딩

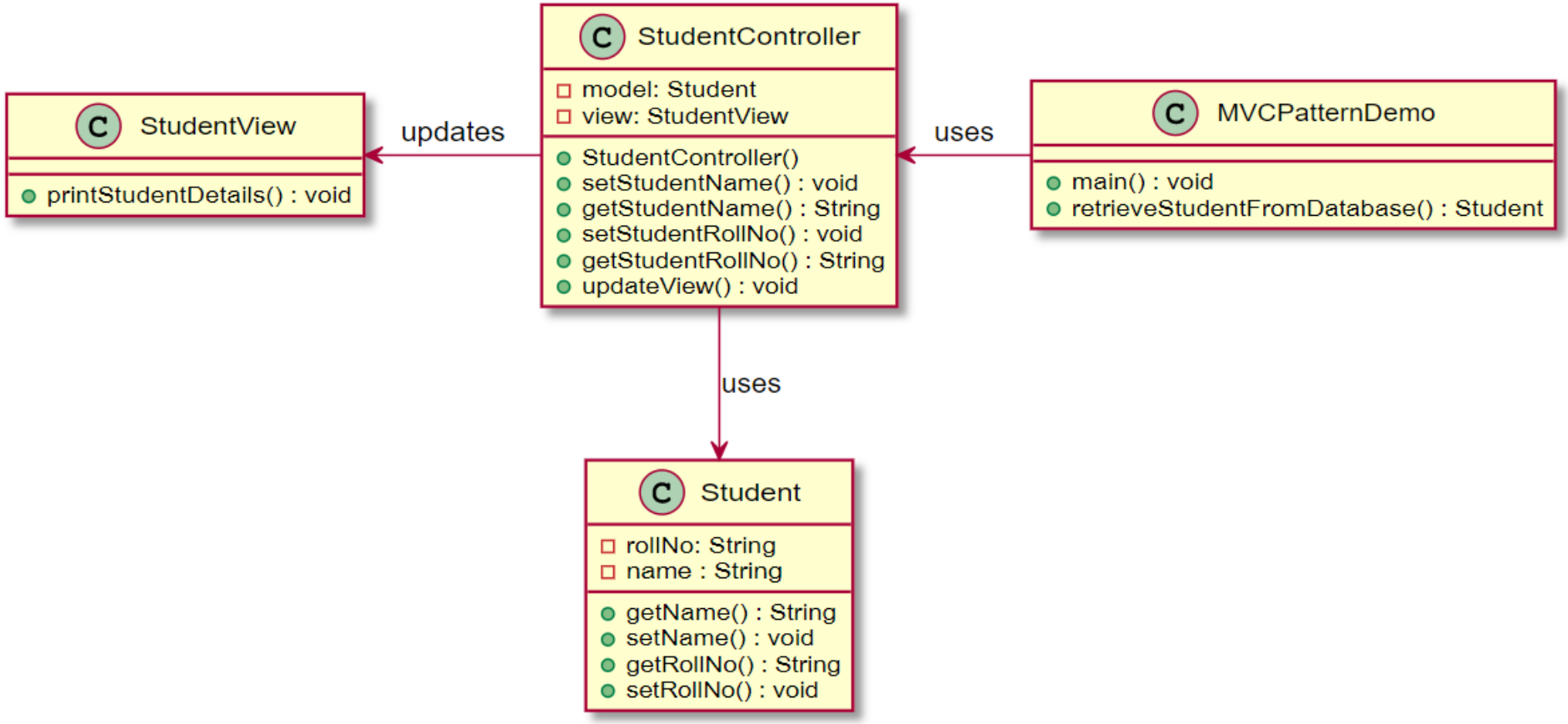

모델
(Model)
업데이트
(updates)
조작 (Manipulates)
뷰
(View)
컨트롤러
(Controller)
보기
(sees)
사용
(uses)
사용자

MVC 패턴의 목적

- 모델과 뷰의 의존성을 제거
  - 모델은 뷰와 상관없이 바뀔 수 있음
  - 뷰도 모델과 상관없이 바뀔 수 있음
- Head First
  - 사용자는 뷰하고만 접촉 가능
    - 뷰가 컨트롤러에게 사용자가 어떤 일을 했는지 알려줌
  - 컨트롤러에서 모델한테 상태를 변경하라는 요청을 함
  - 컨트롤러에서 뷰를 변경해달라고 요청할 수 있음
  - 상태가 변경되면 모델에서 뷰에게 그 사실을 알림
  - 뷰에서 모델한테 상태를 요청

사례

- https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm



사례

```
public class Student {
   private String rollNo;
   private String name;
   public String getRollNo() {
      return rollNo;
   }
   public void setRollNo(String rollNo) {
      this.rollNo = rollNo;
   }
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
}
```

사례

```
public class StudentView {
  public void printStudentDetails(String studentName,
String studentRollNo){
    System.out.println("Student: ");
    System.out.println("Name: " + studentName);
    System.out.println("Roll No: " + studentRollNo);
  }
}
```

사례

```
public class StudentController {
   private Student model;
   private StudentView view;
   public StudentController(Student model,
                            StudentView view) {
      this.model = model;
      this.view = view;
   }
   public void setStudentName(String name) {
      model.setName(name);
   }
   public String getStudentName() {
      return model.getName();
   }
```

사례

```
public void setStudentRollNo(String rollNo) {
    model.setRollNo(rollNo);
  }
  public String getStudentRollNo() {
    return model.getRollNo();
  }
  public void updateView() {
    view.printStudentDetails(model.getName(),
                             model.getRollNo());
  }
}
```

사례

```
public class MVCPatternDemo {
   public static void main(String[] args) {
      // fetch student record based on his roll no
      // from the database
      Student model  = retriveStudentFromDatabase();

      // Create a view : to write student details on
      // console
      StudentView view = new StudentView();
      StudentController controller
                  = new StudentController(model, view);
      controller.updateView();
      //update model data
      controller.setStudentName("John");
      controller.updateView();
   }
```

사례

```
   private static Student retriveStudentFromDatabase() {
     Student student = new Student();
     student.setName("Robert");
     student.setRollNo("10");
     return student;
   }
}
```

MVC 패턴의 장단점

- 장점
  - 클래스간 정보 공유를 최소화시키는 객체 지향 구조
    - 모델과 뷰가 서로에 대해서 잘 몰라도 됨
  - 같은 모델에서 여러 개의 뷰를 지원할 수 있음
- 단점
  - 비효율적일 수 있음
    - 뷰가 업데이트 되어야 함을 알려주면, 뷰는 모델로부터 정보를 받아서 업데이트함
    - 모델이 뷰가 필요로 하는 것을 직접 전달하는 것이 더 효율적이지만, 객체 지향적이지는 않음
    - 컨트롤러의 역할이 너무 커질 수 있음

MVC 모델 2 (웹)