Comp 401
Model-View-Controller (MVC)

Instructor: Prasun Dewan

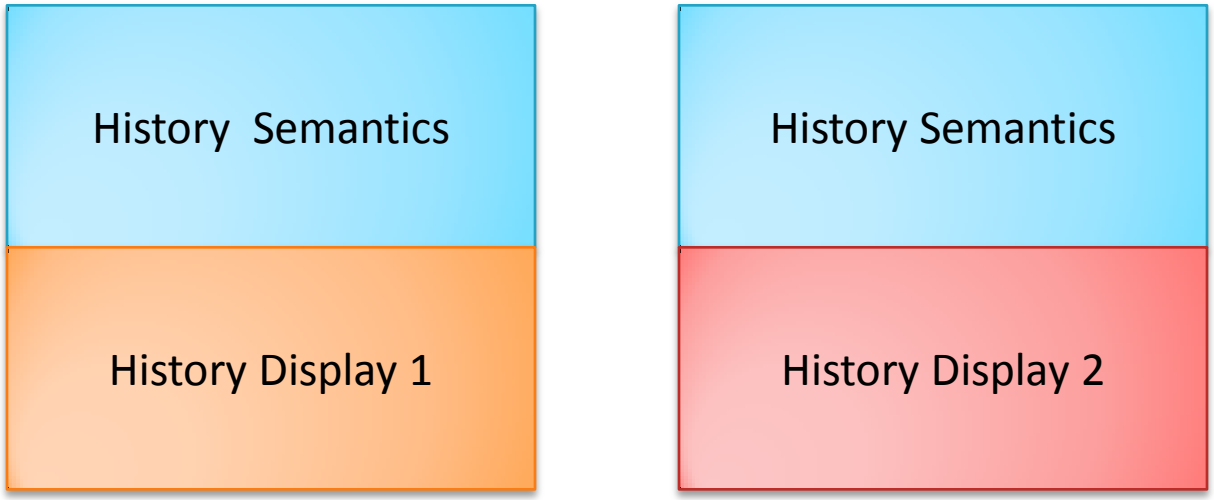
Prerequisites

- Interfaces
- Main Console Input
- Inheritance

General Problem

- How to break up our program into multiple classes?

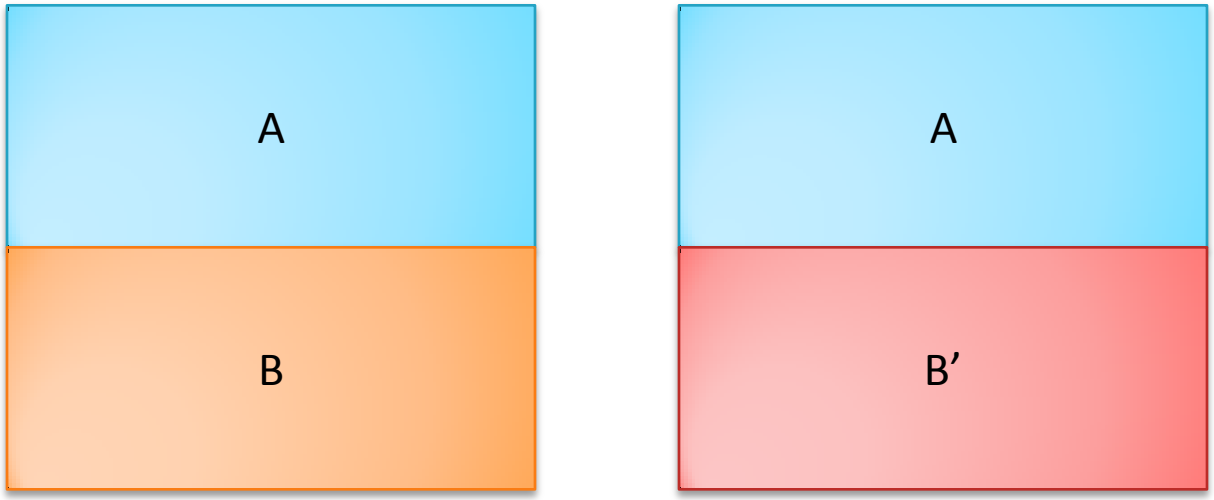
Separation of Concerns



Can change display without changing other aspects of history

Display and semantics should go in different classes

Separation of Concerns

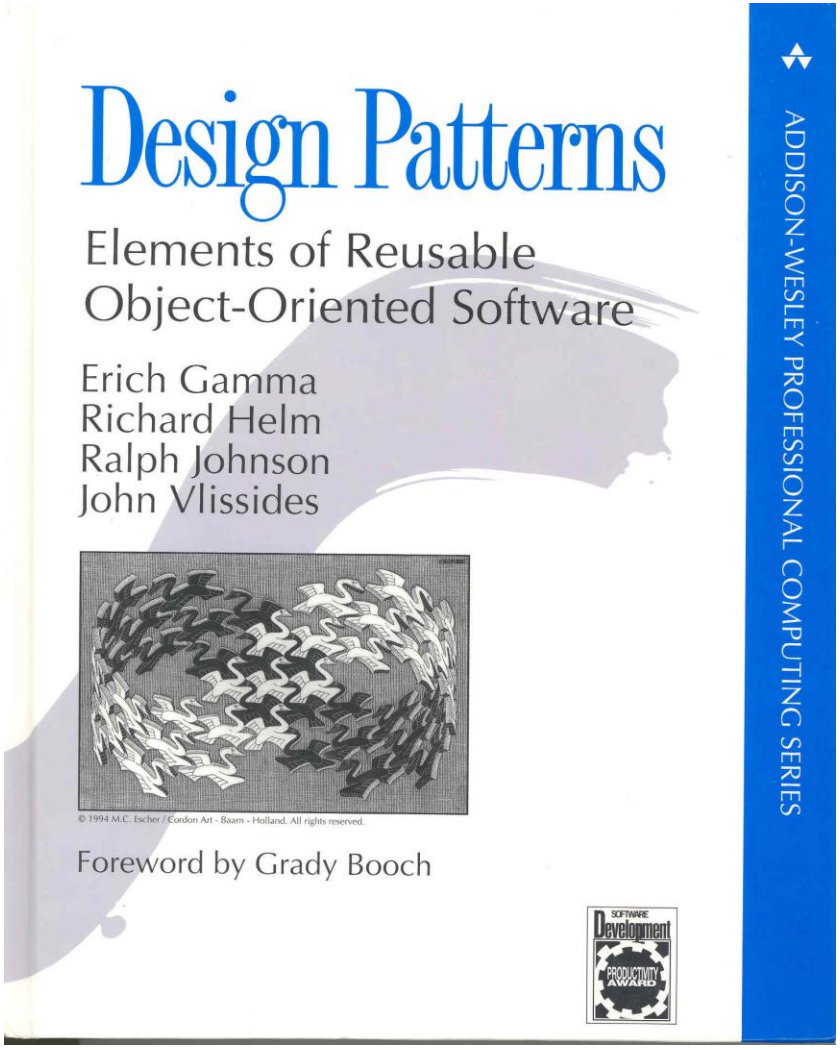


if a part A of a class can be changed without changing some other part B of the class, then refactor and put A and B in different classes

Patterns

- Recurring theme
- Bean, Vector pattern
  - Conventions for readability
- Loop patterns
  - Event-controlled
  - Counter controlled
- Design patterns
  - Helps identify the kind of classes our program should have

Design Pattern



- Reusable program decomposition pattern
- Not specific class or interface, infinite family of classes/interfaces implement this pattern.
- Usually involves multiple objects
- Language-independent
- Include architecture and frameworks
- Inspired by Architectural Pattern (Chrstopher Plummer)

SIMILAR

2nd story porch supported by 1st floor porch columns

DIFFERENT

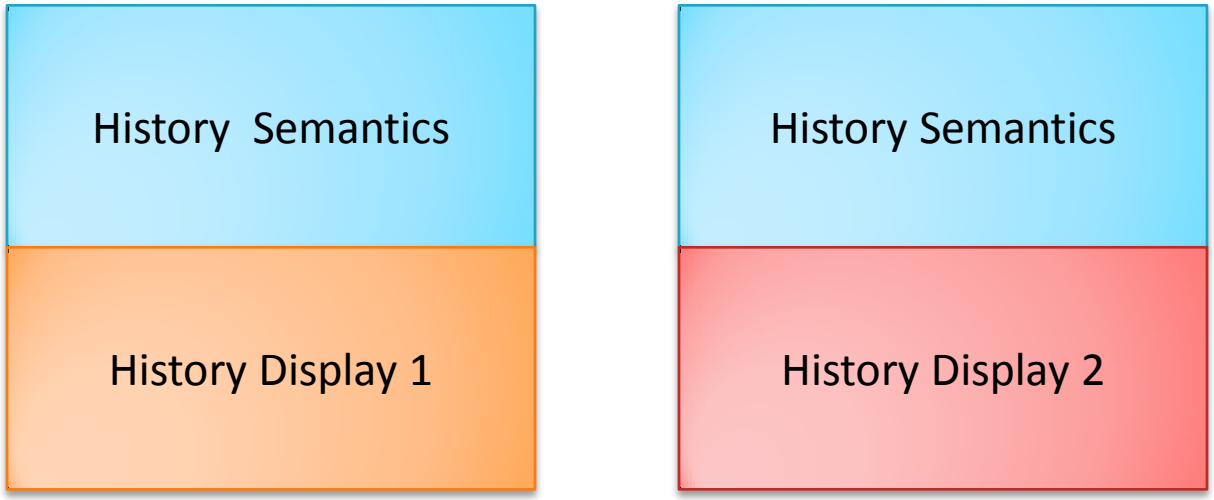
No outside stairs to 2nd story

Flat (not bay) window

Wide plank siding

Screened-in porch

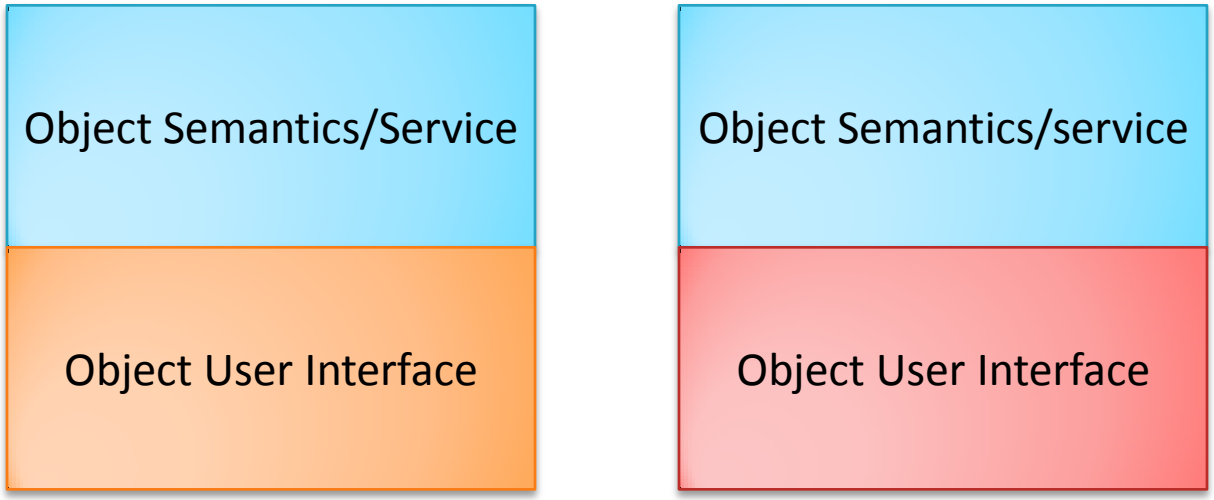
MVC Motivation



Can change display without changing other aspects of history

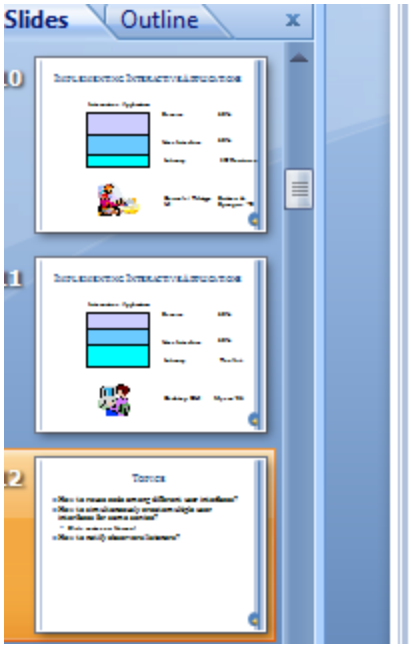
Display and semantics should go in different classes

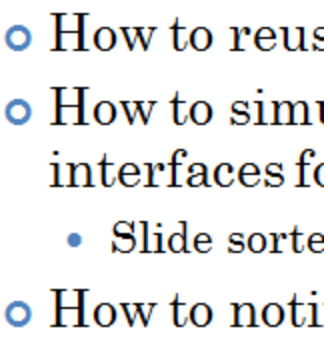
MVC Motivation

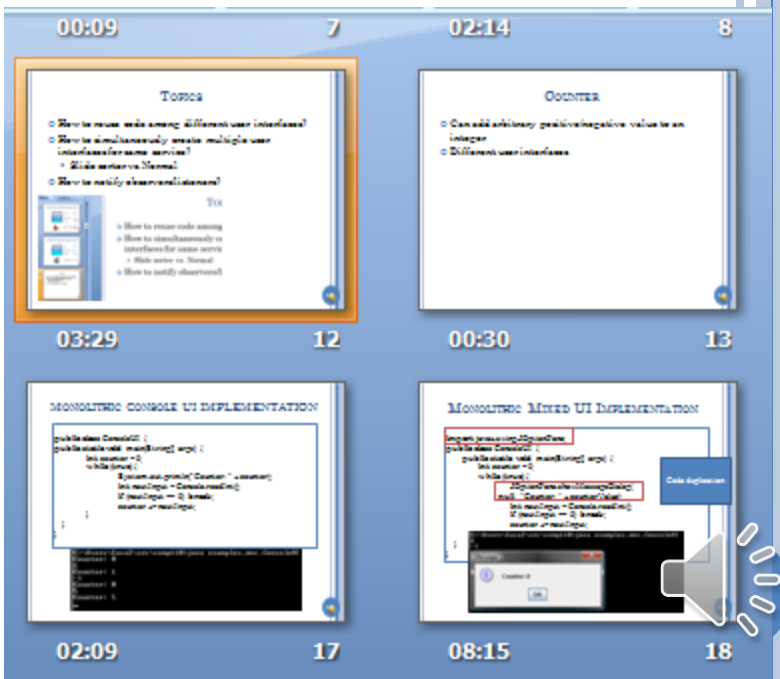


Questions

- How to reuse code among different interactive applications offering the same service?
- How to simultaneously create multiple user interfaces for same service?
  - Normal vs. Slide sorter
  - Shortcuts vs. menus vs. buttons





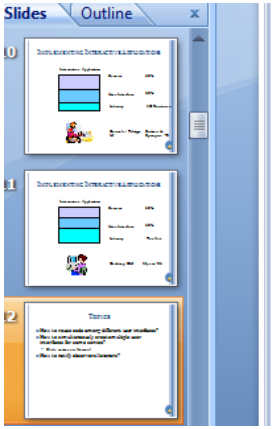


Questions

- How to simultaneously create multiple user interfaces for same service on different computers?
  - Facebook, email

Questions

- How to simultaneously create distributed user interfaces
  - multiple complete user interfaces for different users on different computers
  - Single user-interface on large computer controlled by multiple mobile devices



- How to reus
- How to simu
  interfaces fo
  - Slide sorte
- How to noti

Example: Counter

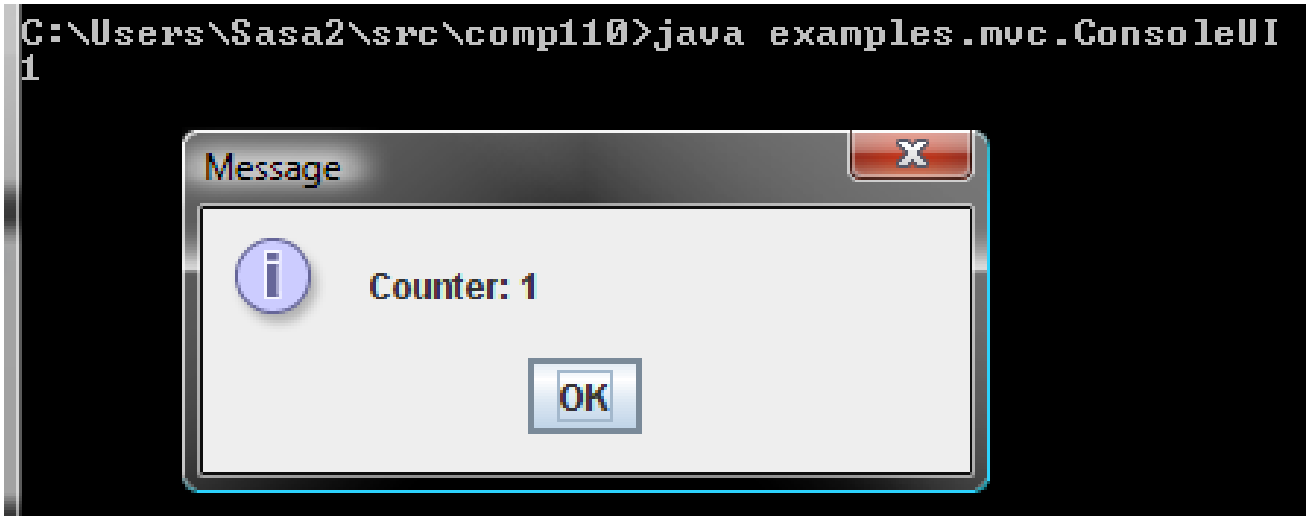
Can add arbitrary positive/negative value to an integer

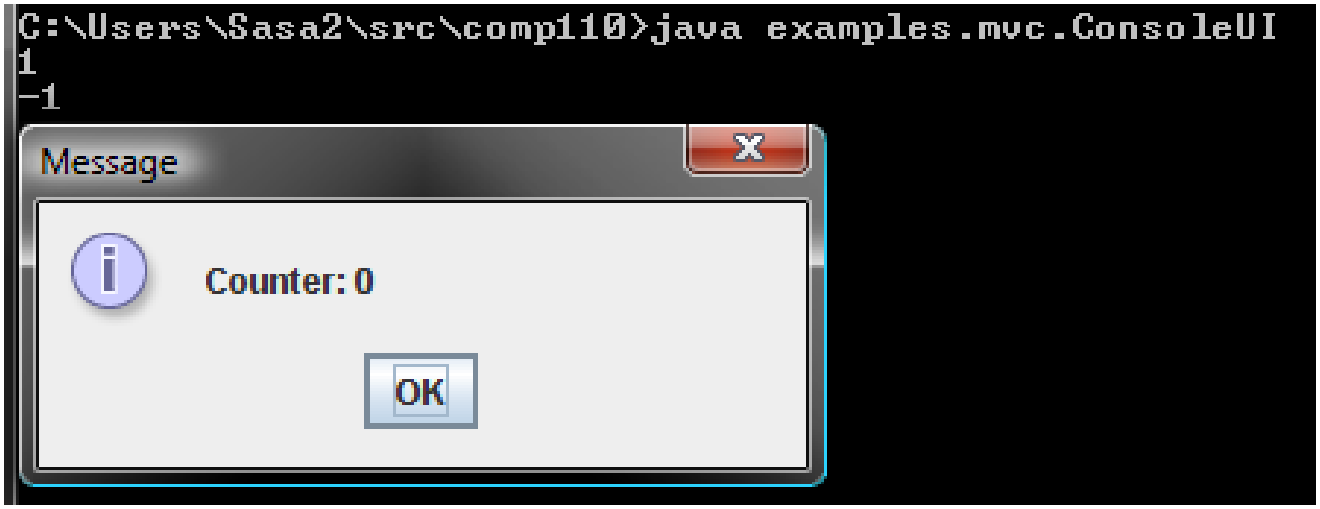
Different user interfaces

Console Input and Output

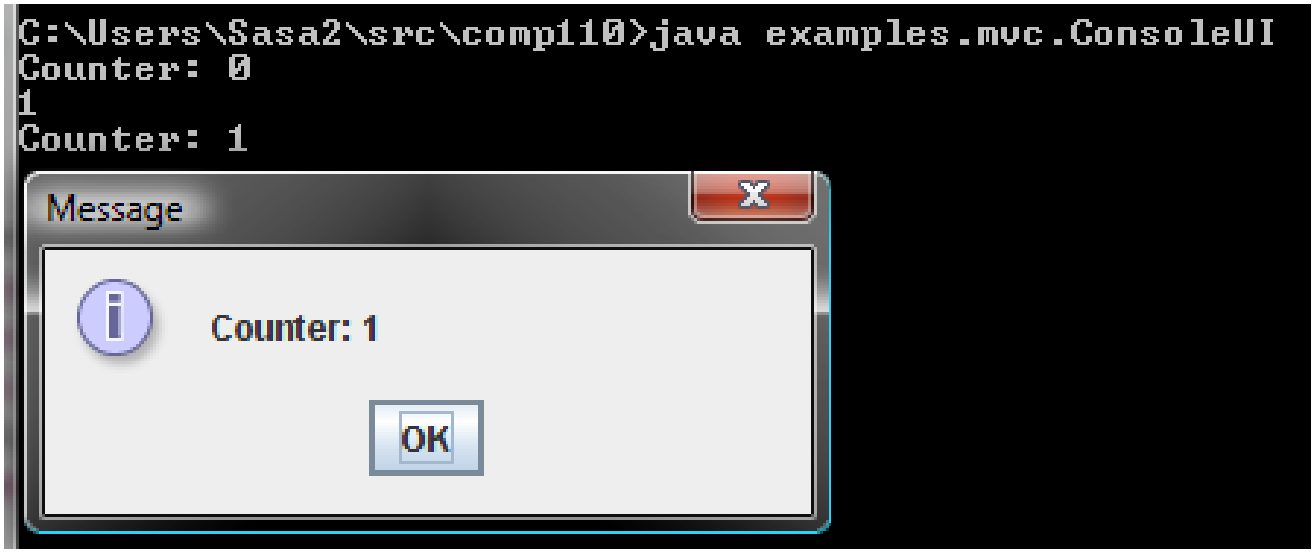
```
C:\Users\Sasa2\src\comp110>java examples.mvc.ConsoleUI
Counter: 0
1
Counter: 1
-1
Counter: 0
5
Counter: 5
```

Console Input and JOption Output





Console Input,Output and JOption Output



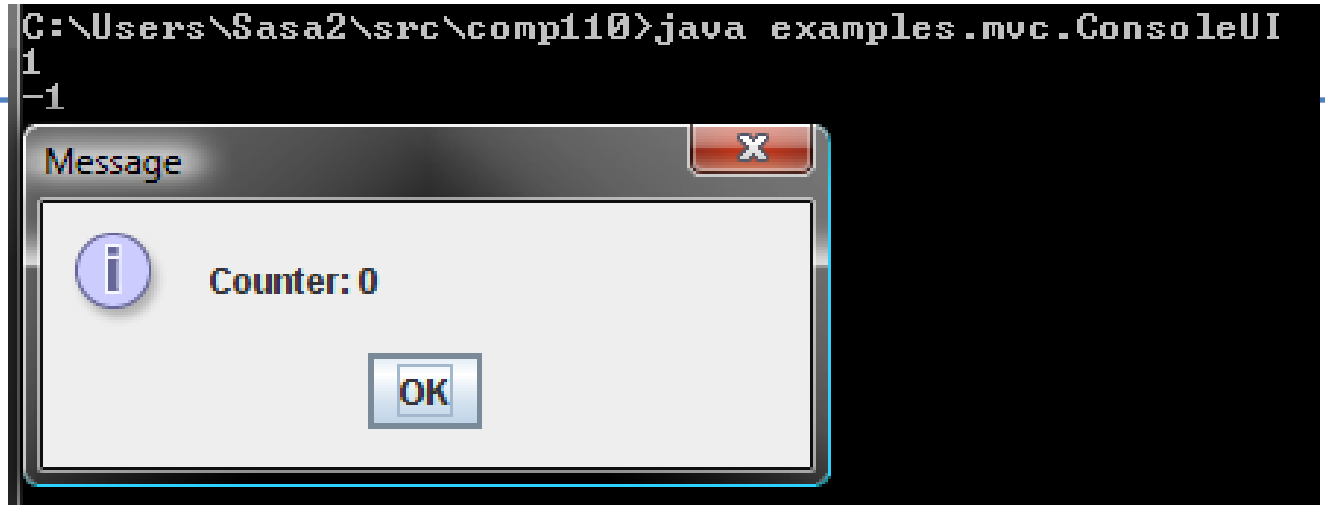
Monolithic Console UI IMPLEMENTATION

```
public class MonolithicConsoleUI {
public static void main(String[] args) {
        int counter = 0;
        while (true) {
                System.out.println("Counter: " + counter);
                int nextInput = Console.readInt();
                if (nextInput == 0) break;
                counter += nextInput;
        }
   }
}
```

```
C:\Users\Sasa2\src\comp110>java examples.mvc.ConsoleUI
Counter: 0
1
Counter: 1
-1
Counter: 0
5
Counter: 5
-
```

Monolithic Console UI Implementation

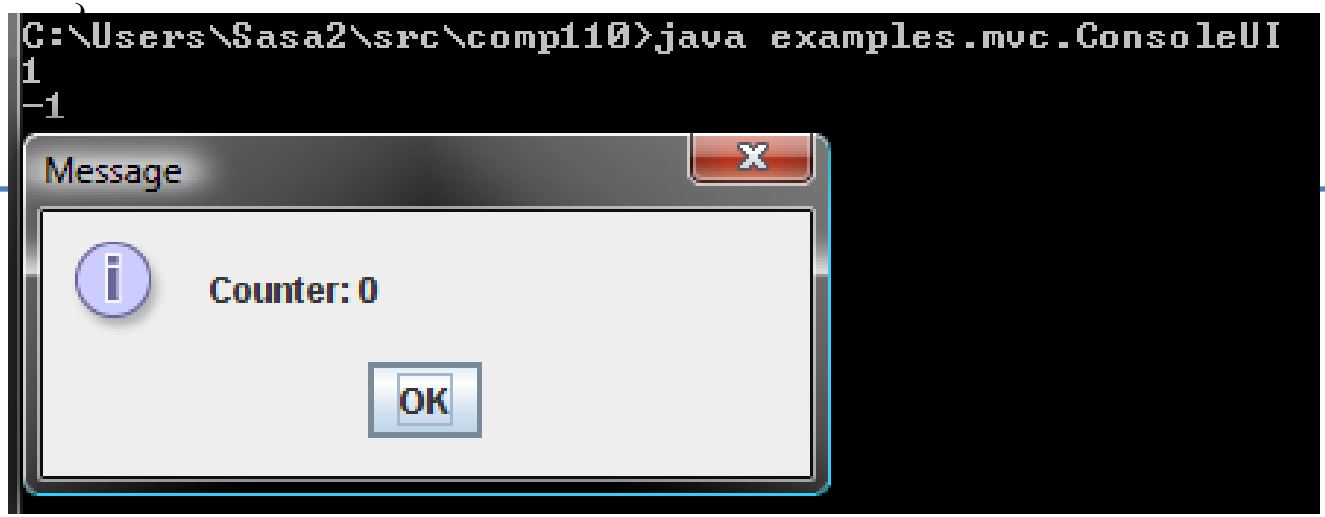
```
public class MonolithicConsoleUI {
public static void main(String[] args) {
        int counter = 0;
        while (true) {
                System.out.println("Counter: " + counter);
                int nextInput = Console.readInt();
                if (nextInput == 0) break;
                counter += nextInput;
        }
   }
}
```



Monolithic Mixed UI Implementation

```
import javax.swing.JOptionPane;
public class ConsoleUI {
        public static void main(String[] args) {
            int counter = 0;
            while (true) {
                    JOptionPane.showMessageDialog(
                  null, "Counter: " + counter);
                    int nextInput = Console.readInt();
                    if (nextInput == 0) break;
                    counter += nextInput;
    }
}
```

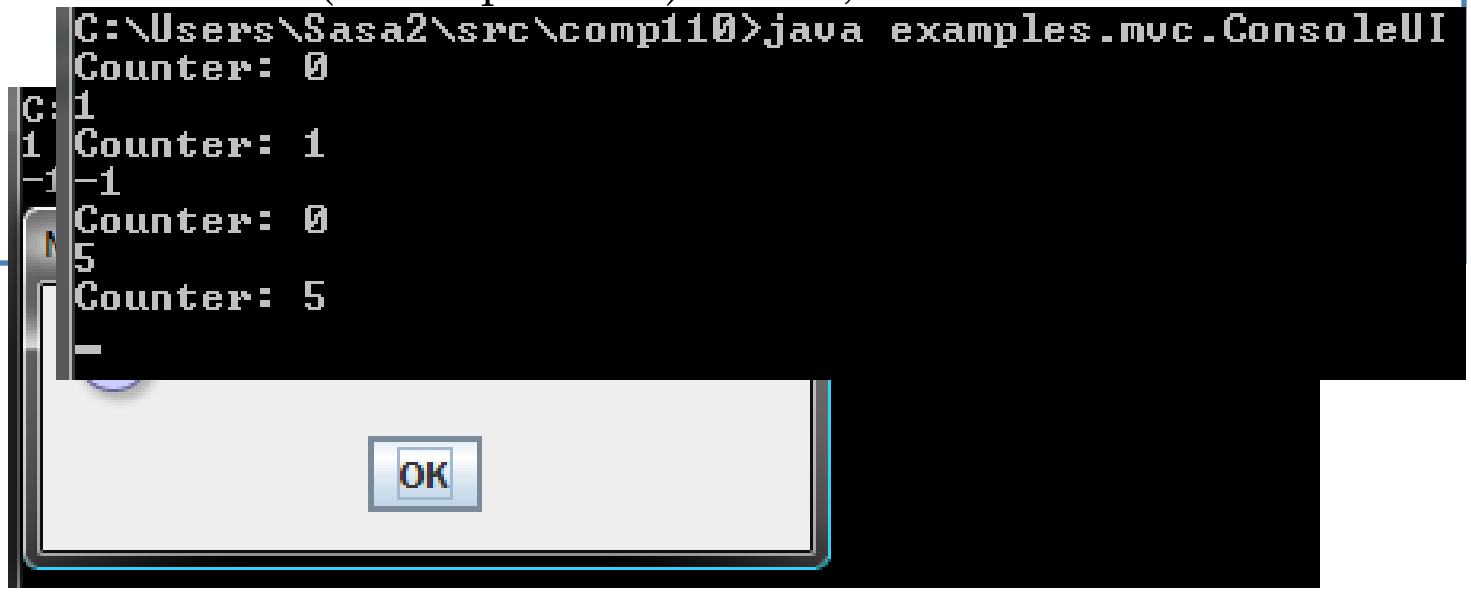
Code duplication



Multiple UIs?

```
import javax.swing.JOptionPane;
public class ConsoleUI {
        public static void main(String[] args) {
                int counter = 0;
                while (true) {
                        JOptionPane.showMessageDialog(
                    null, "Counter: " + counter);
                        int nextInput = Console.readInt();
                        if (nextInput == 0) break;
    }
}
```

Cannot use both UIs simultaneously to update same counter

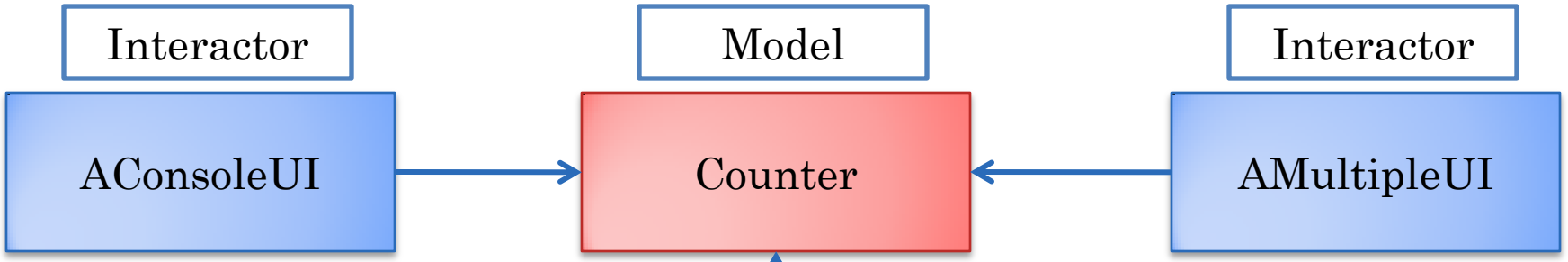


Counter Model

```
public class ACounter implements Counter {
        int counter = 0;
        public void add (int amount) {
                counter += amount;
        }
        public int getValue() {
                return counter;
        }
}
```

No input/output

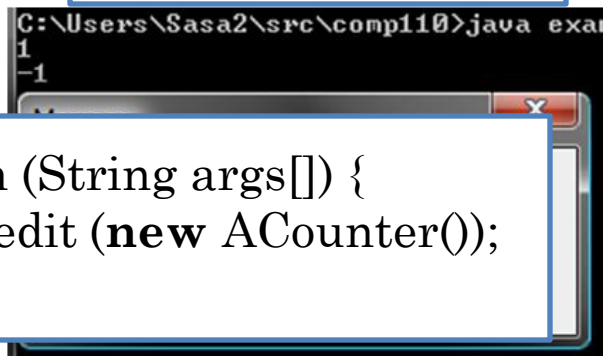
Model/Interactor(Editor) Separation

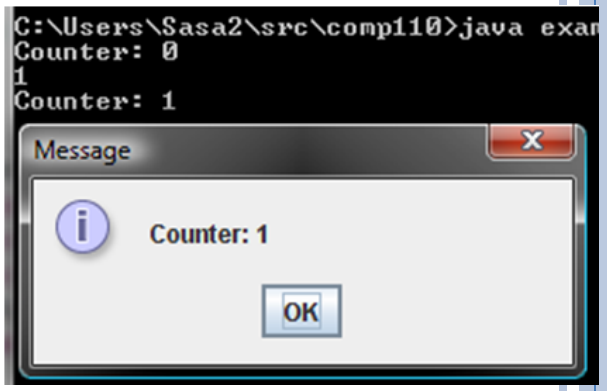



```
C:\Users\Sasa2\
Counter: 0
1
Counter: 1
-1
Counter: 0
5
Counter: 5
```

AMixedUI

Interactor or Editor

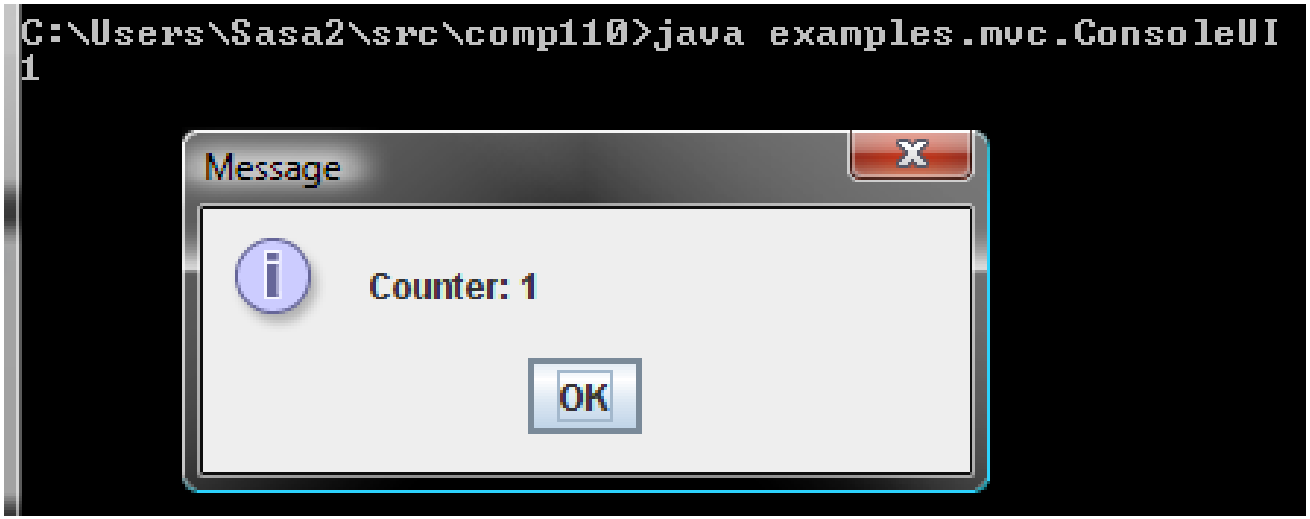





```
public static void main (String args[]) {
    (new AConsoleUI()).edit (new ACounter());
}
```

Model has no UI code and only semantics!

Model?


C:\Users\Sasa2\src\comp110>java examples.mvc.ConsoleUI
1
Message
Counter: 1
OK

Counter Model

```
public class ACounter implements Counter {
        int counter = 0;
        public void add (int amount) {
                counter += amount;
        }
        public int getValue() {
                return counter;
        }
}
```

No input/output

Monolithic Console UI Implementation

```
public class ConsoleUI {
public static void main(String[] args) {
        int counter = 0;
        while (true) {
                System.out.println("Counter: " + counter);
                int nextInput = Console.readInt();
                if (nextInput == 0) break;
                counter += nextInput;
        }
    }
}
```

Console Interactor

```
public class AConsoleUIInteractor implements CounterInteractor {
        public void edit (Counter counter) {
            while (true) {
                System.out.println("Counter: " + counter.getValue());
                int nextInput = Console.readInt();
                if (nextInput == 0) return;
                counter.add(nextInput);
            }
    }
}
```

Mixed Interactor

```
public class AMixedUIInteractor implements CounterInteractor {
        public void edit (Counter counter) {
                while (true) {
                        JOptionPane.showMessageDialog(null,
                                "Counter: " + counter.getValue());
                        int nextInput = Console.readInt();
                        if (nextInput == 0) break;
                        counter.add(nextInput);
                }
        }
}
```

Shared Model Code

Output

Input

I/O Code is Duplicated

UI Implementation is now monolithic

Multiple UI Interactor

```
public class AMultipleUI implements CounterInteractor {
        public void edit (Counter counter) {
                while (true) {
                        System.out.println("Counter: " + counter.getValue());
                        JOptionPane.showMessageDialog(null,
                                "Counter: " + counter.getValue());
                        int nextInput = Console.readInt();
                        if (nextInput == 0) break;
                        counter.add(nextInput);
                }
        }
}
```

Shared Model Code

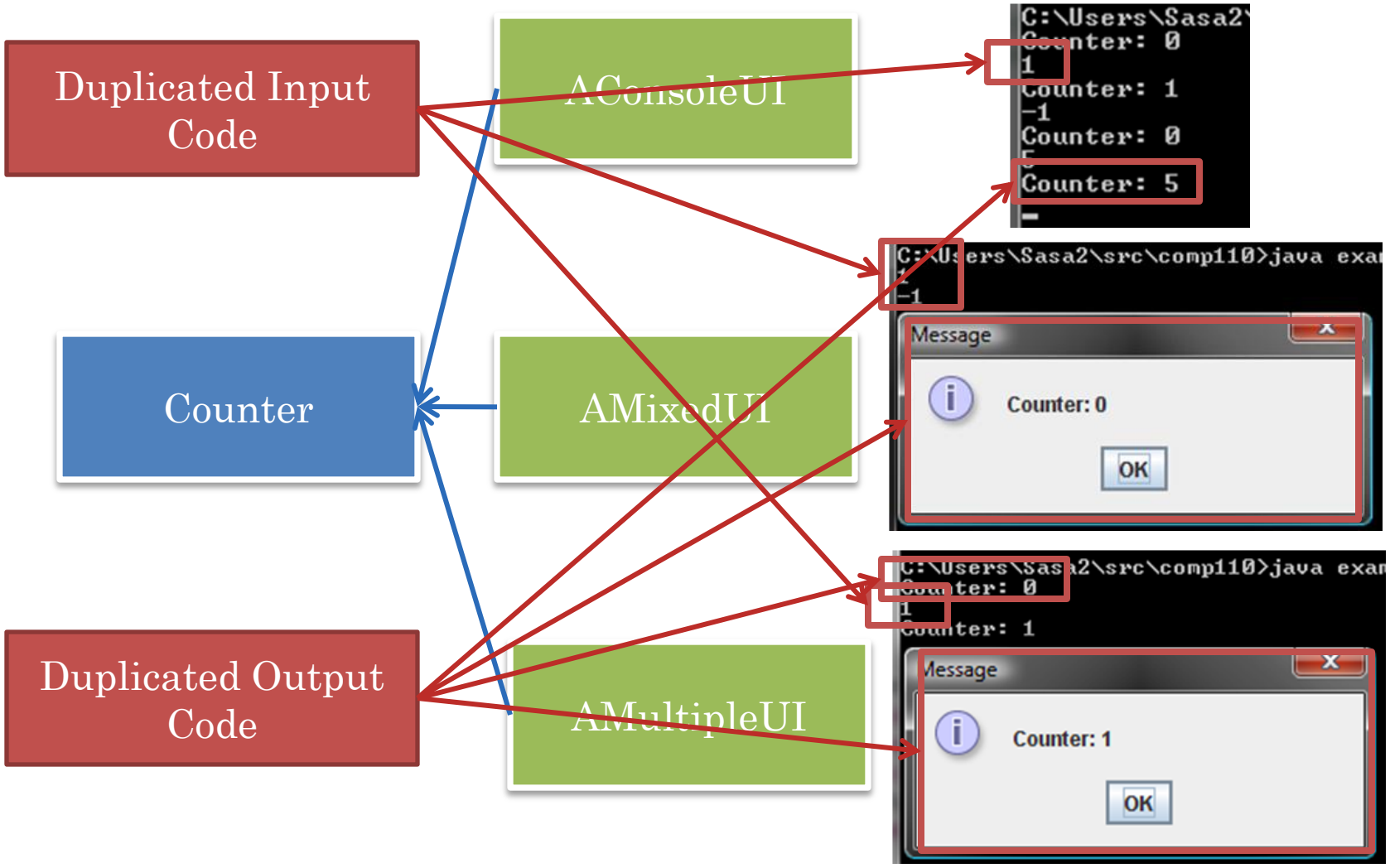
Output

Input

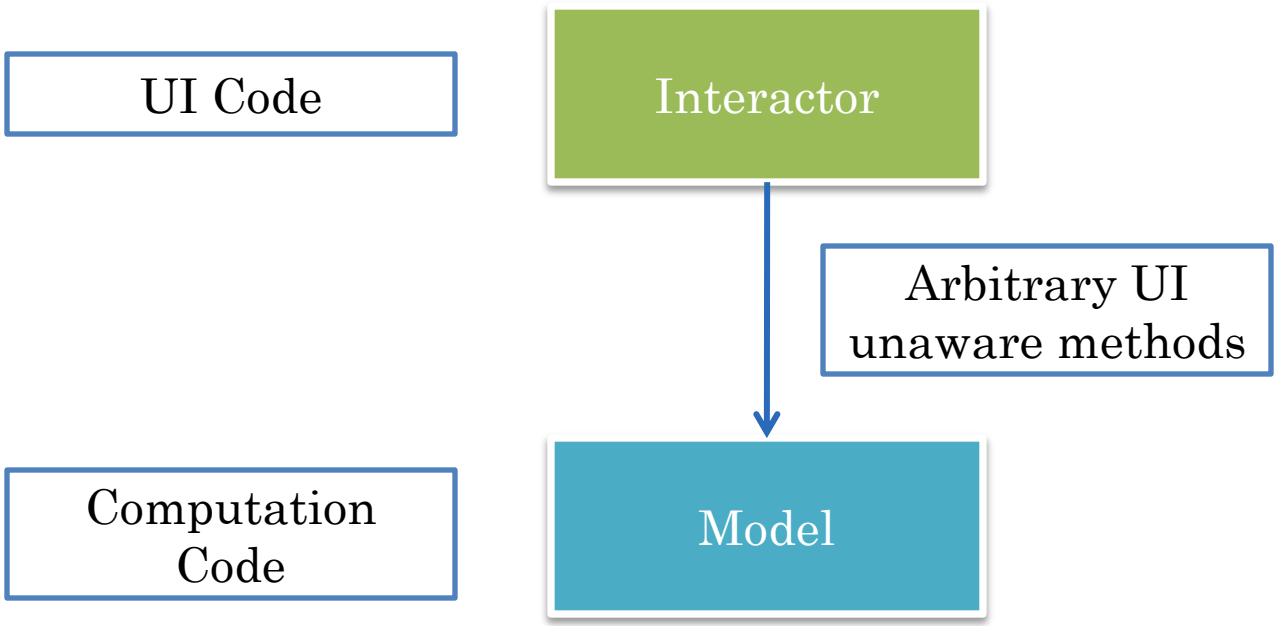
I/O Code is Duplicated

UI Implementation is now monolithic

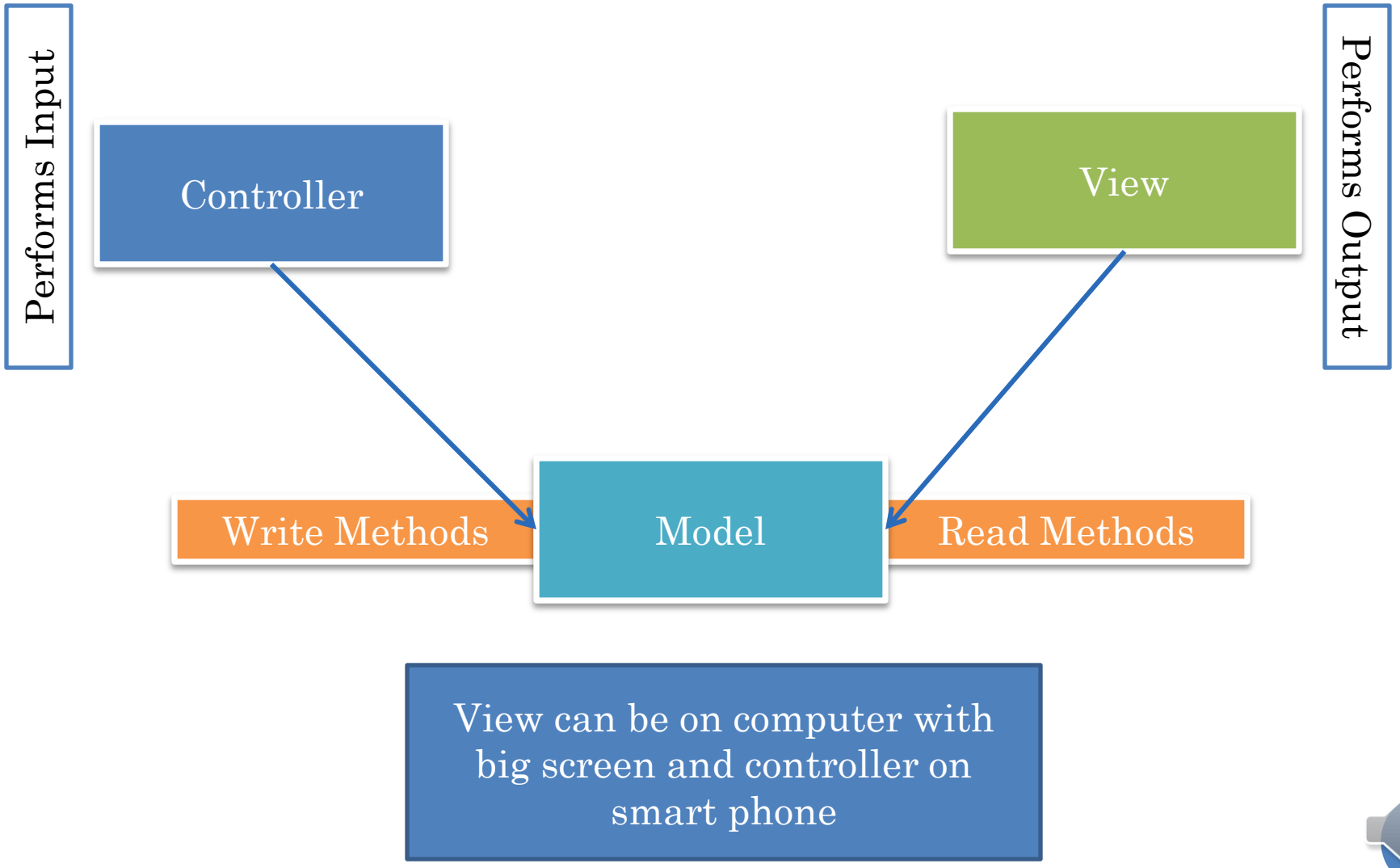
Drawbacks of Monolithic UI



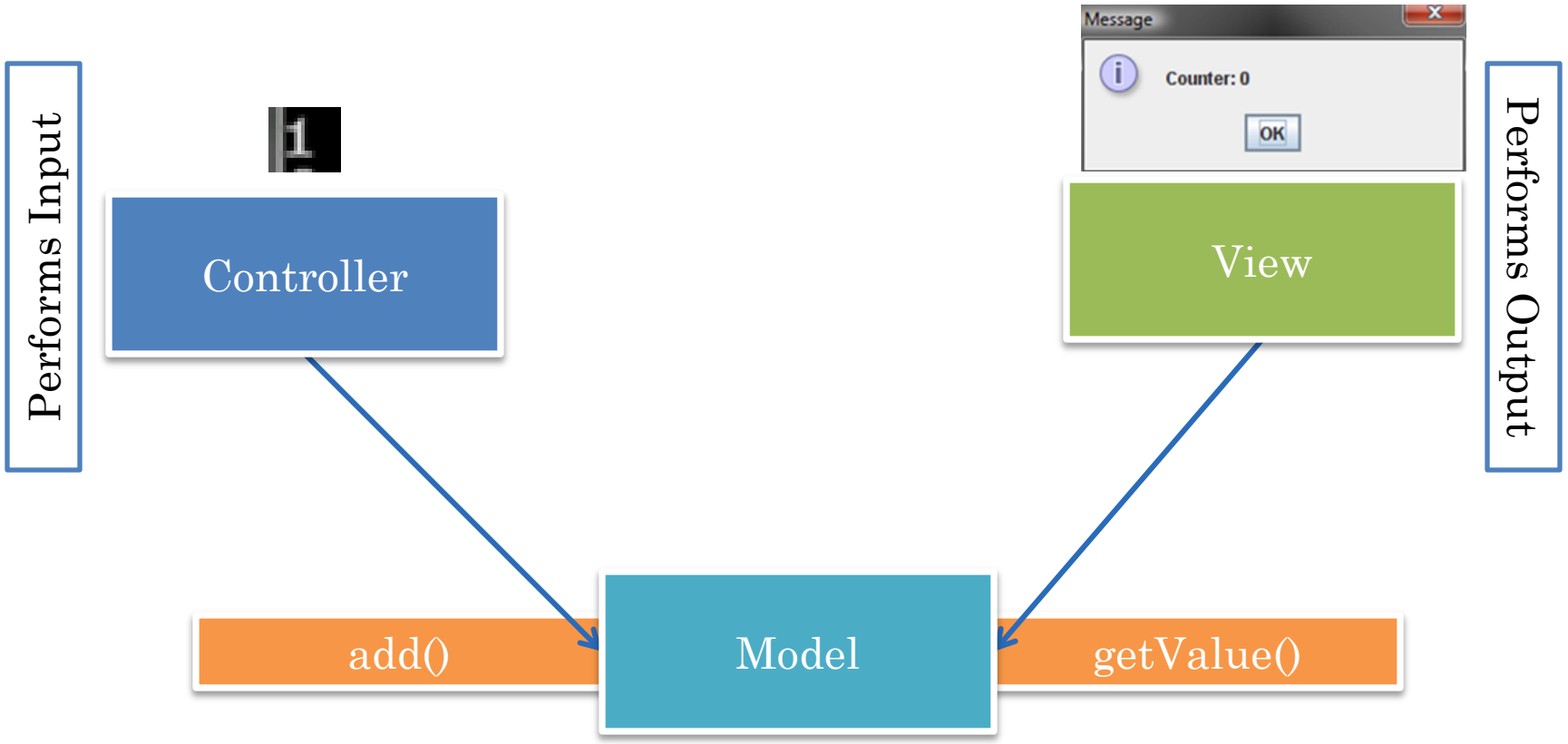
Model/Interactor Pattern



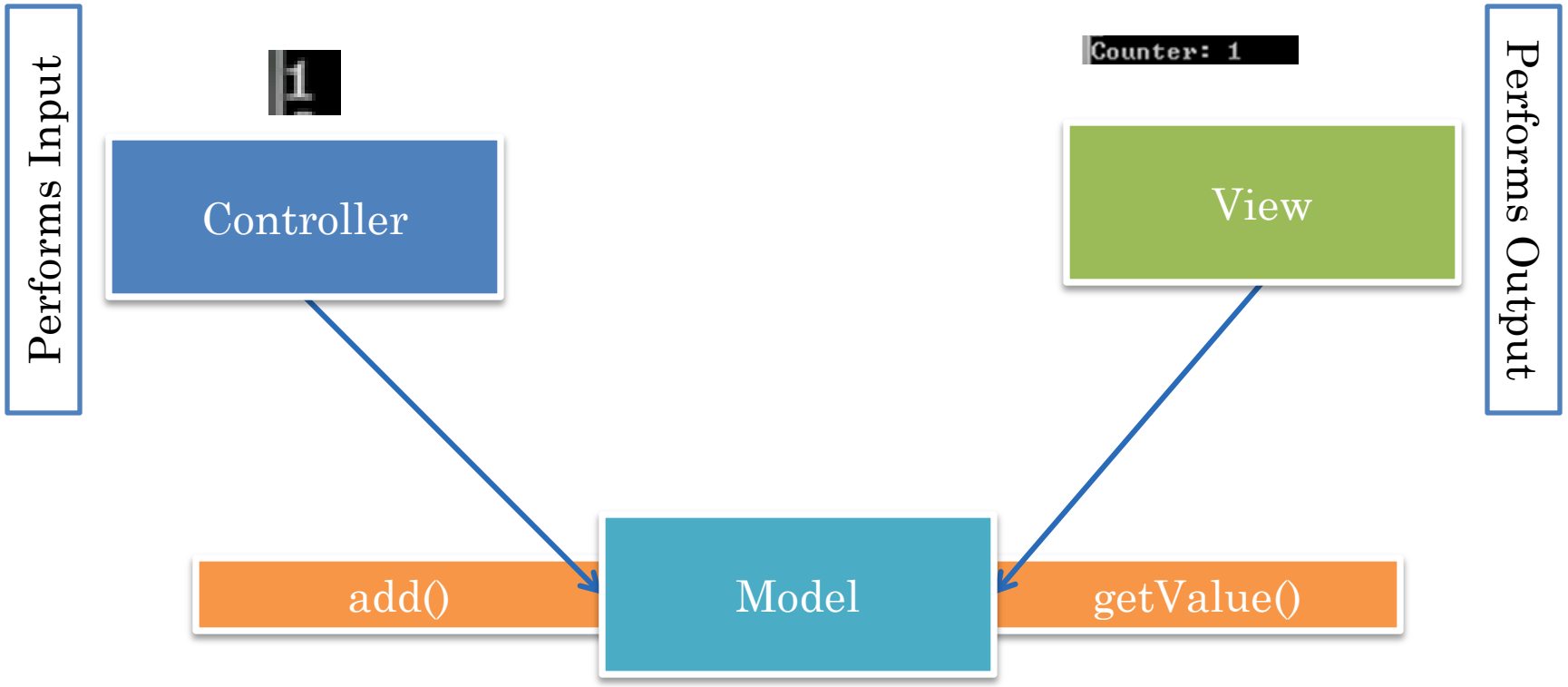
MVC Pattern


Performs Input
Controller
View
Performs Output
Write Methods
Model
Read Methods
View can be on computer with big screen and controller on smart phone

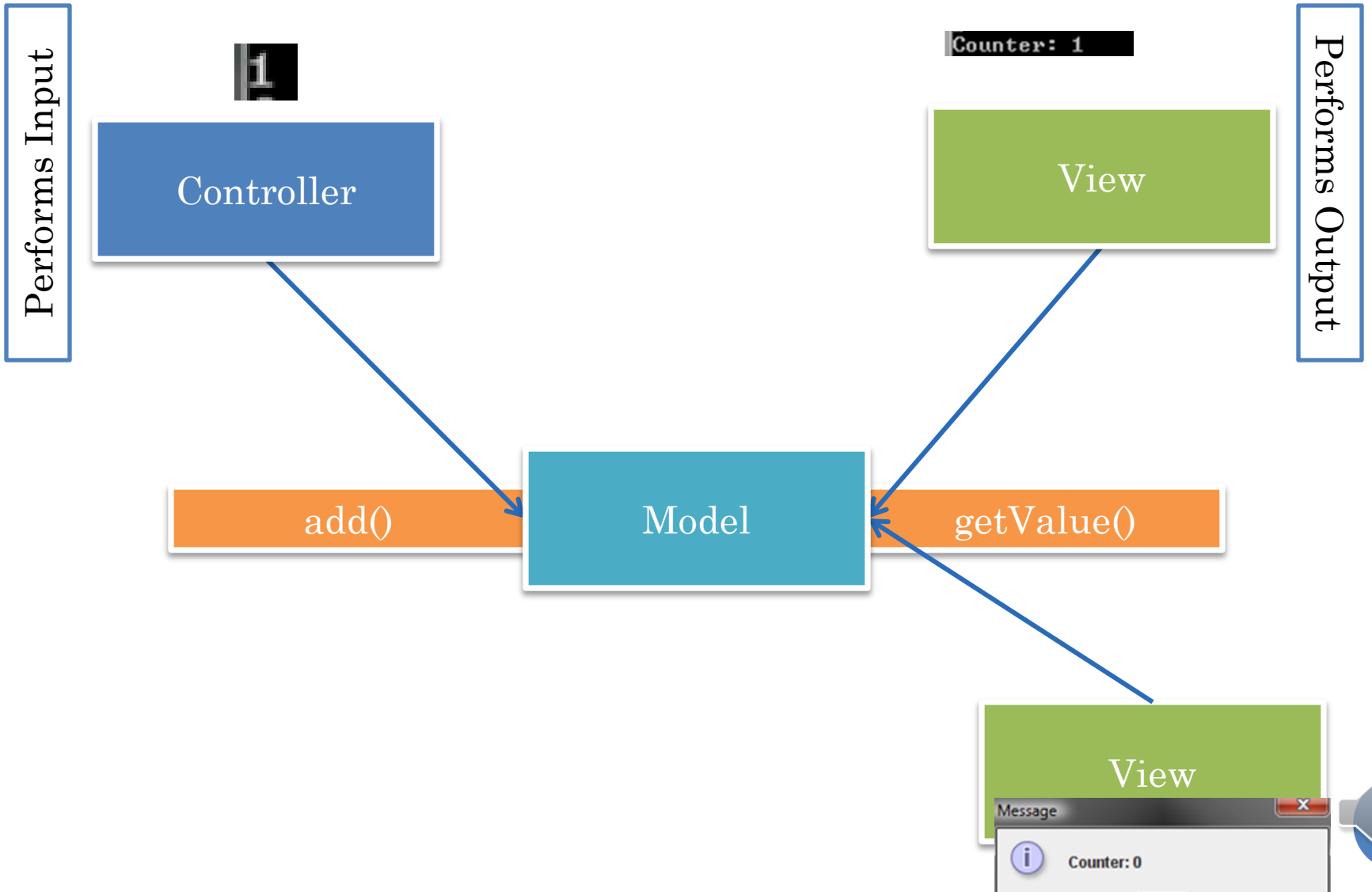
MVC Pattern in Counter



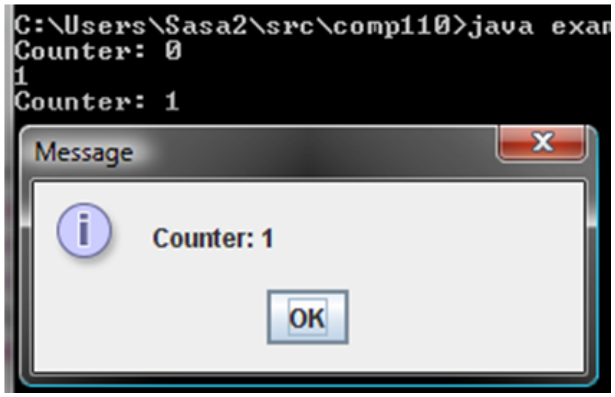
Changing to Console View



Multiple Views



Multiple Views and Controllers



Controller 1

Controller 2

Controller 3

Controller 4

Model

View 1

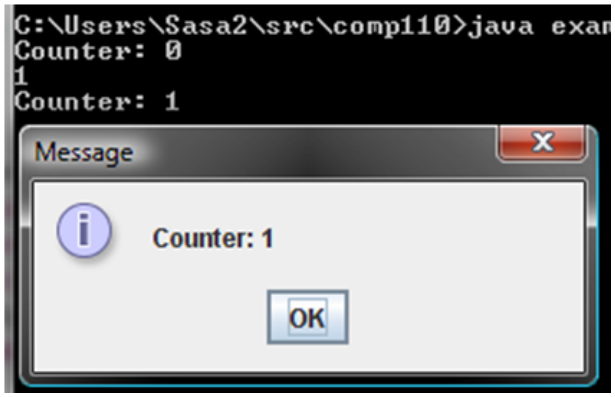
View 2

View 3

View 4

Menus, buttons, shortcuts can be handled by different controllers (in same or different program)

Syncing Controllers & View



Controller 1

Controller 2

Controller 3

Controller 4

Model

View 1

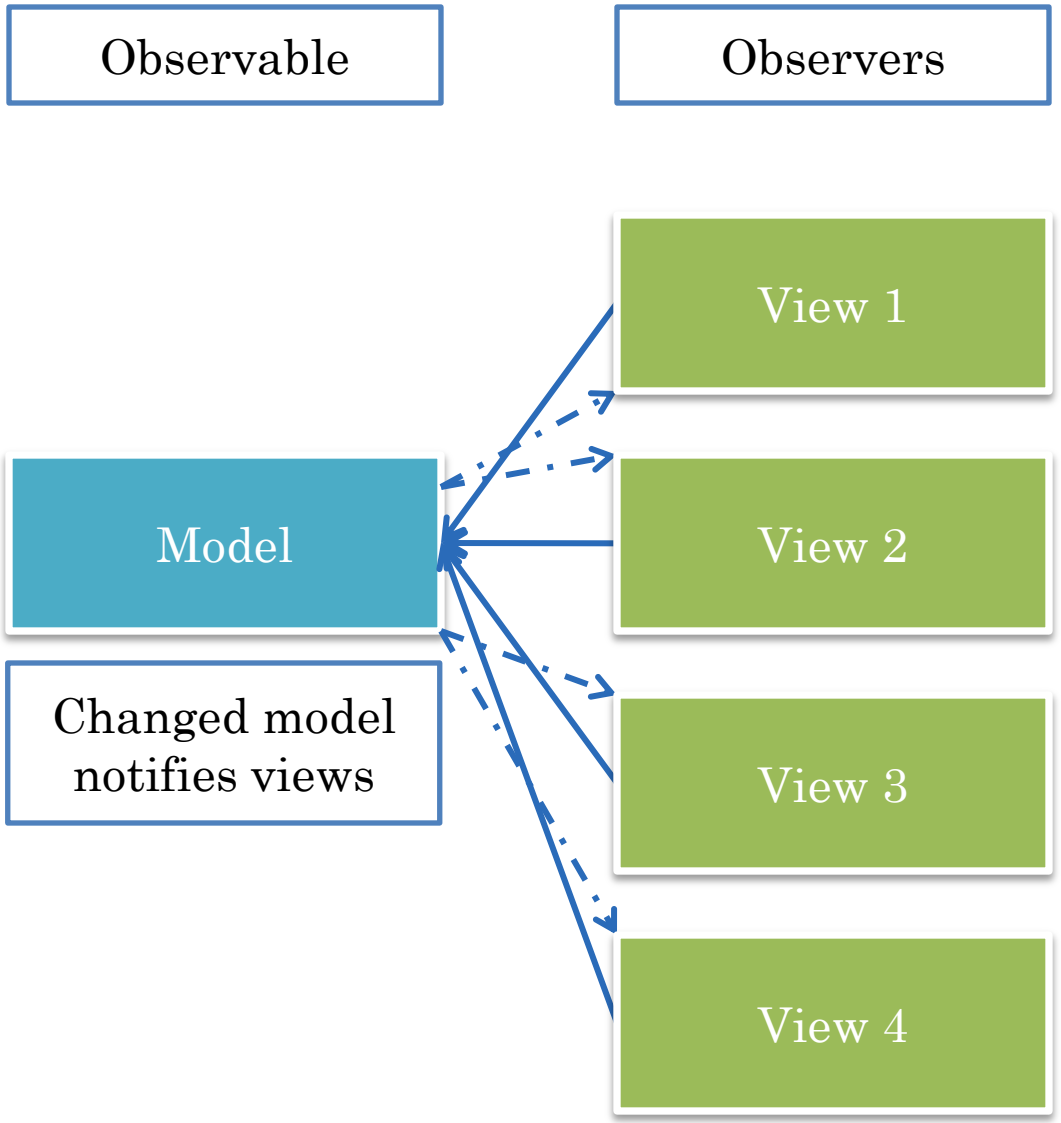
View 2

View 3

View 4

In Http-based "MVC" a single view and controller exist in the browser and the model in the server. A Model cannot initiate actions in the browser so the controller directly communicates with the view

Observer Pattern

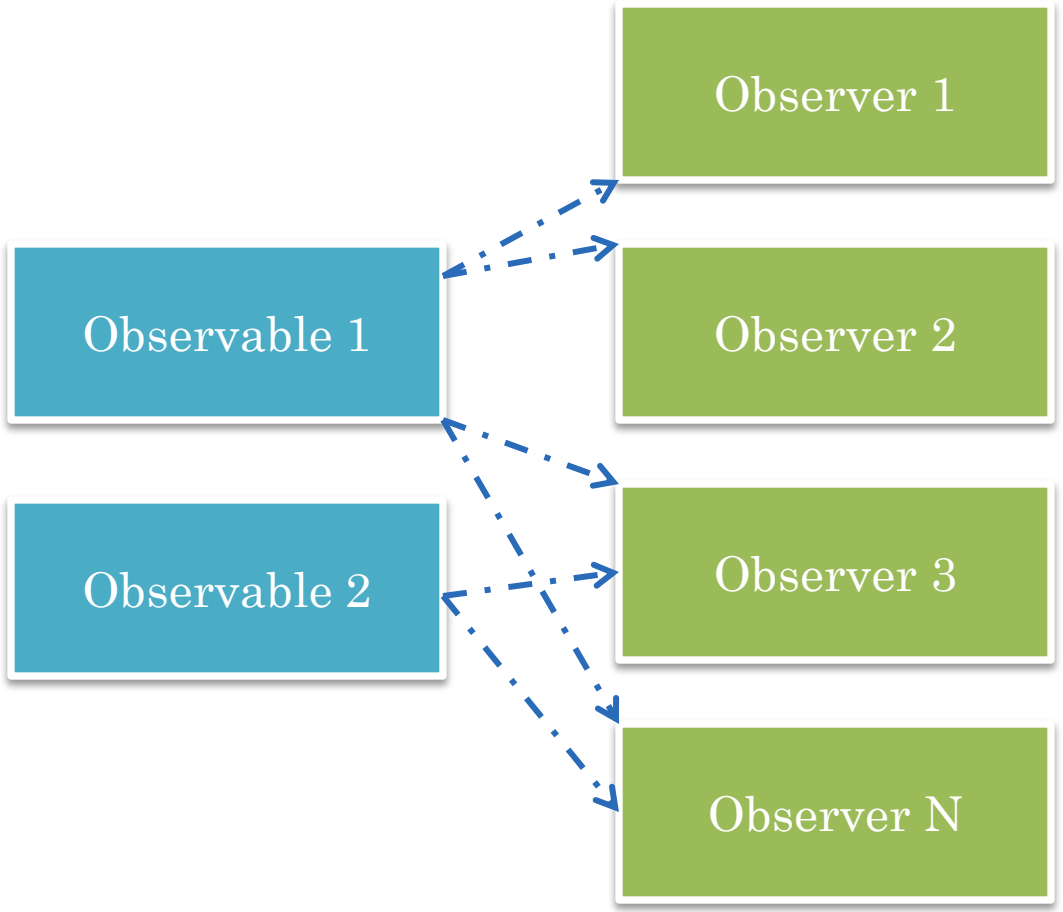


Multiple Observers/Observables

- A single battle simulation view observing
  - Multiple planes
  - Multiple tanks

How does observable know about its observers?

Observer registered with observable



Notification Scheme

Each observer is registered with observable

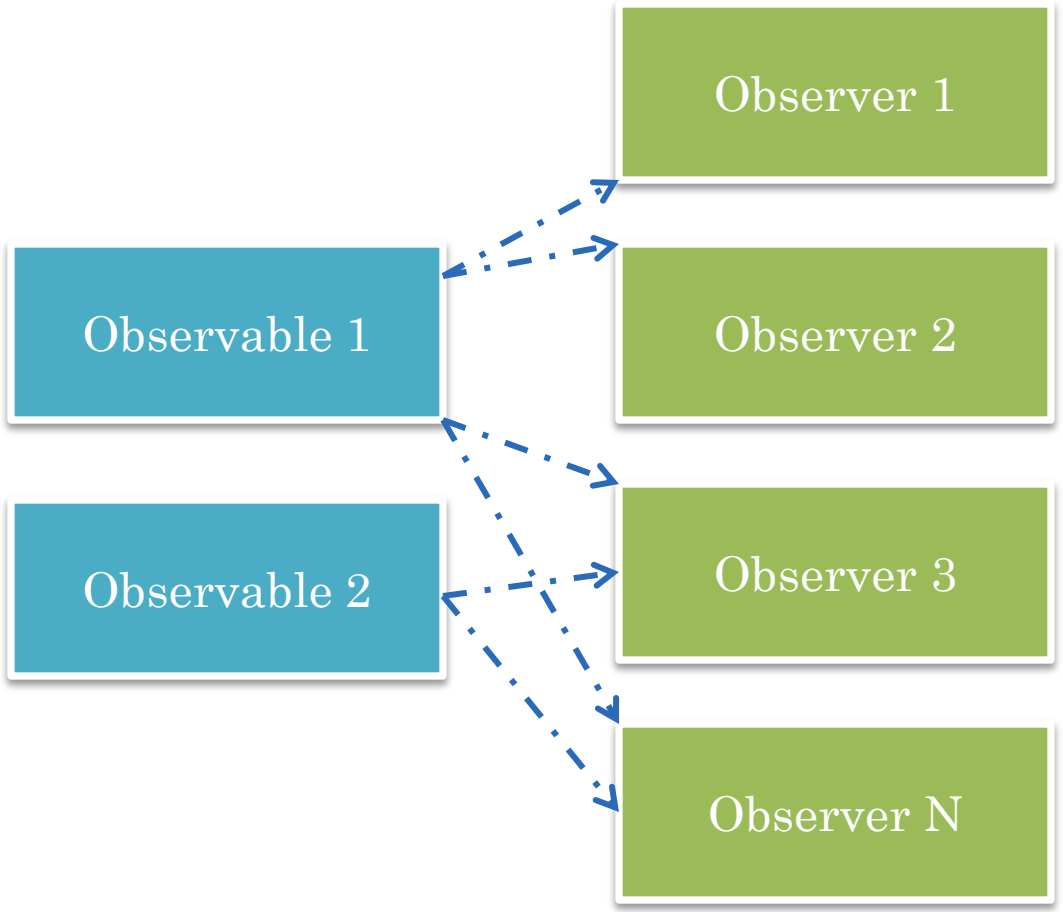
Each write method in observable calls a notification method in each observer

Notification method in observer reads model

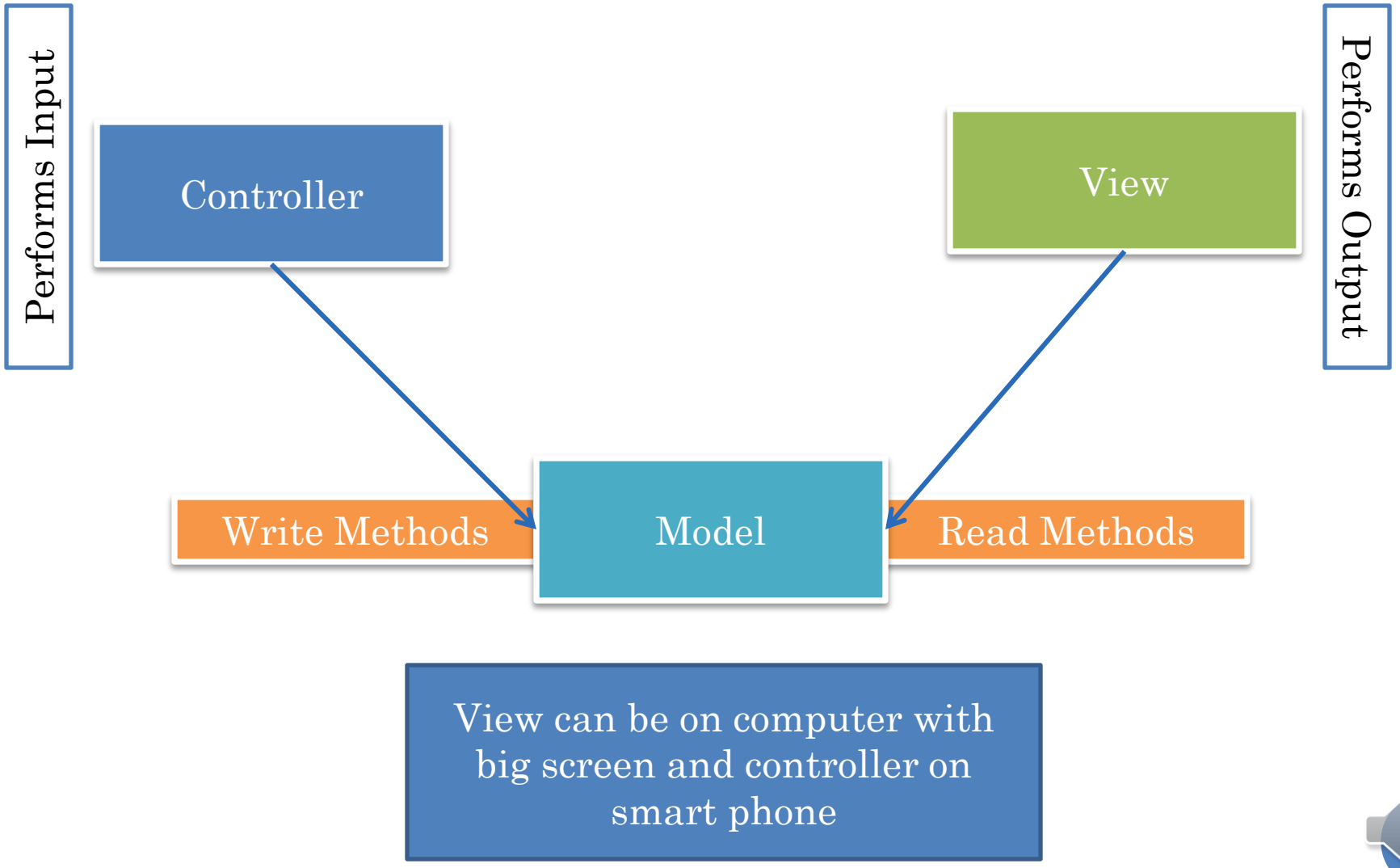
Each student is registered with professor's listserv

When web page is updated mail sent to students

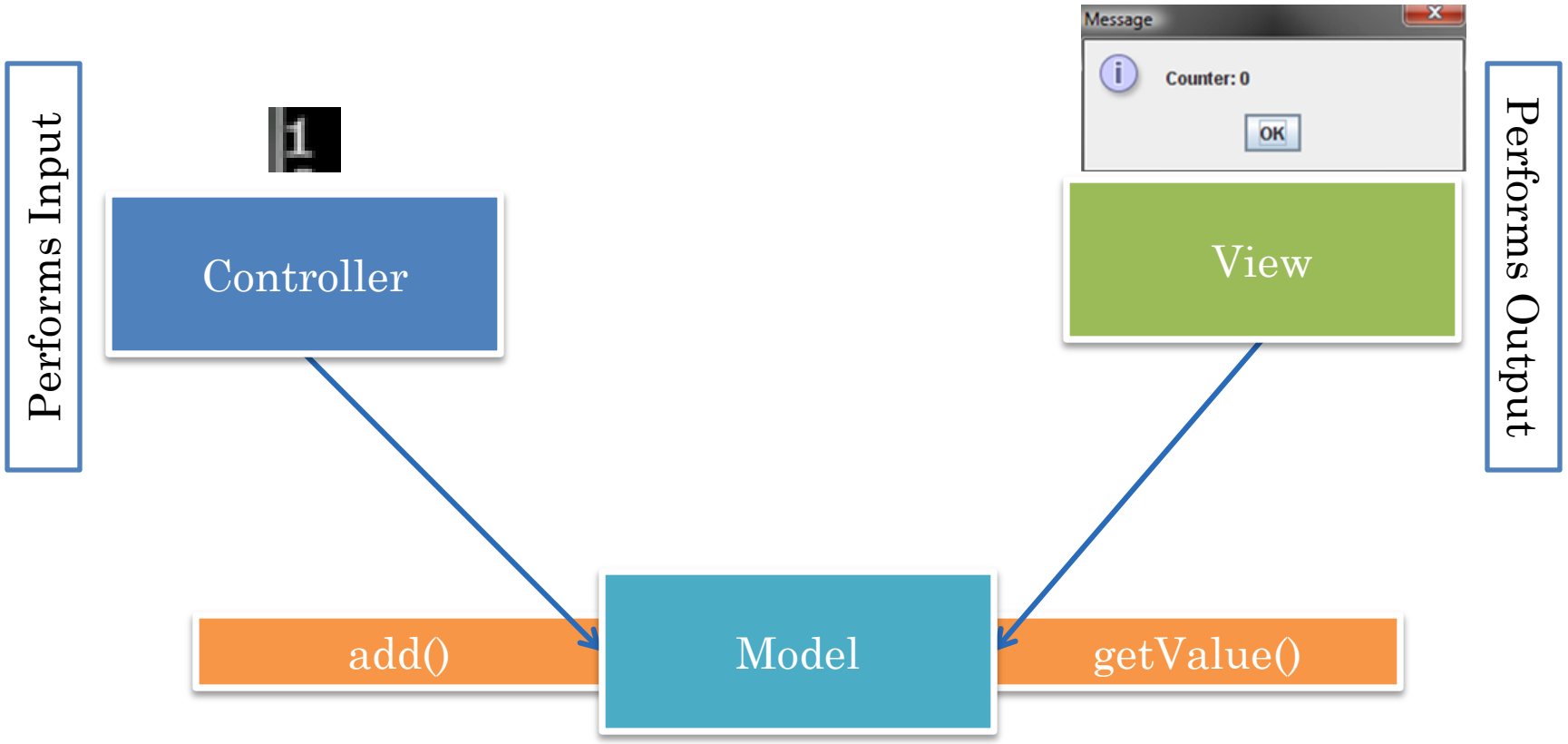
Student reads web page if mailed information is not sufficient



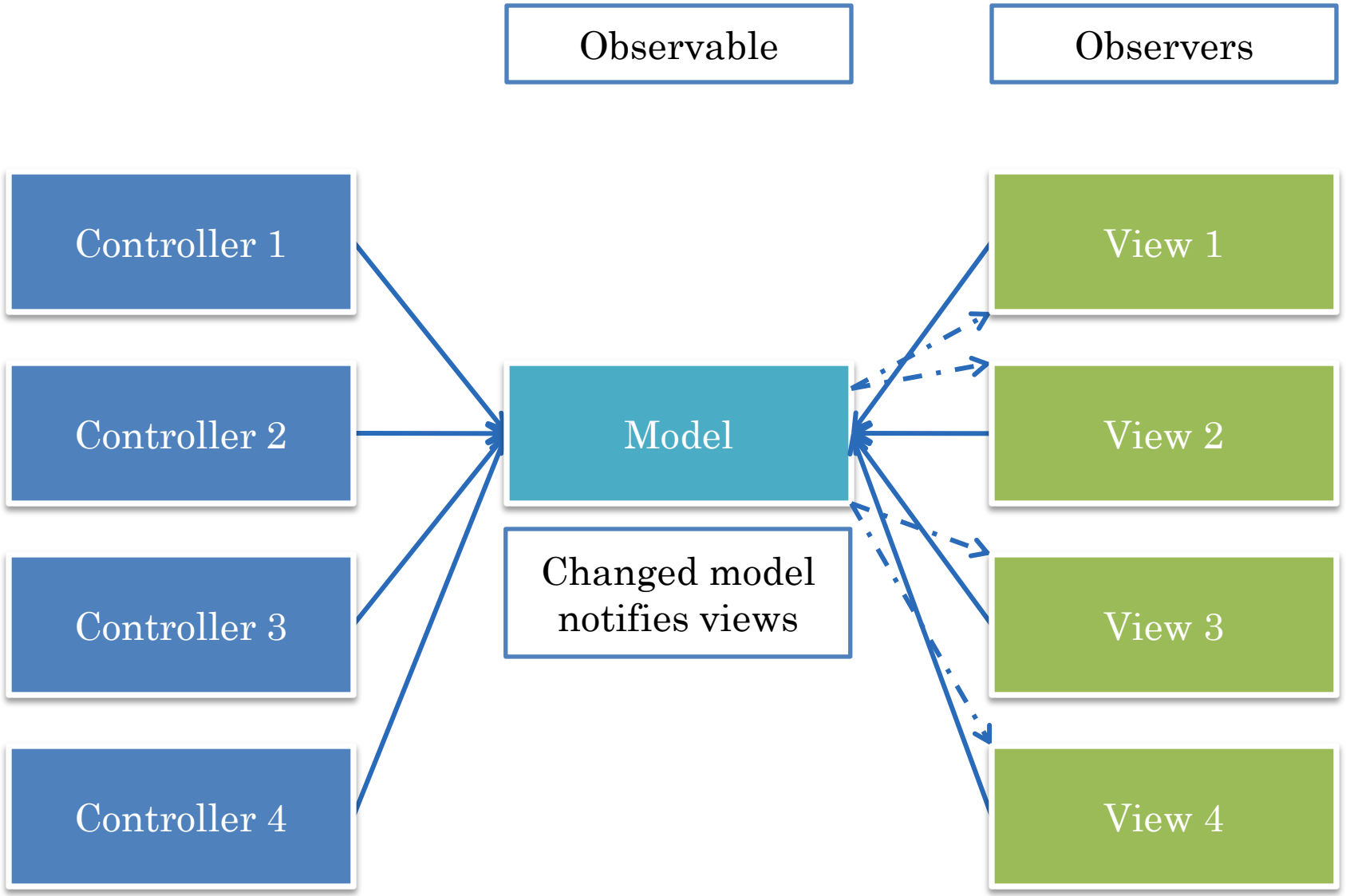
MVC Pattern (Review)



MVC Pattern in Counter (Review)



Observer Pattern



Notification Scheme (Review)

Each observer is registered with observable

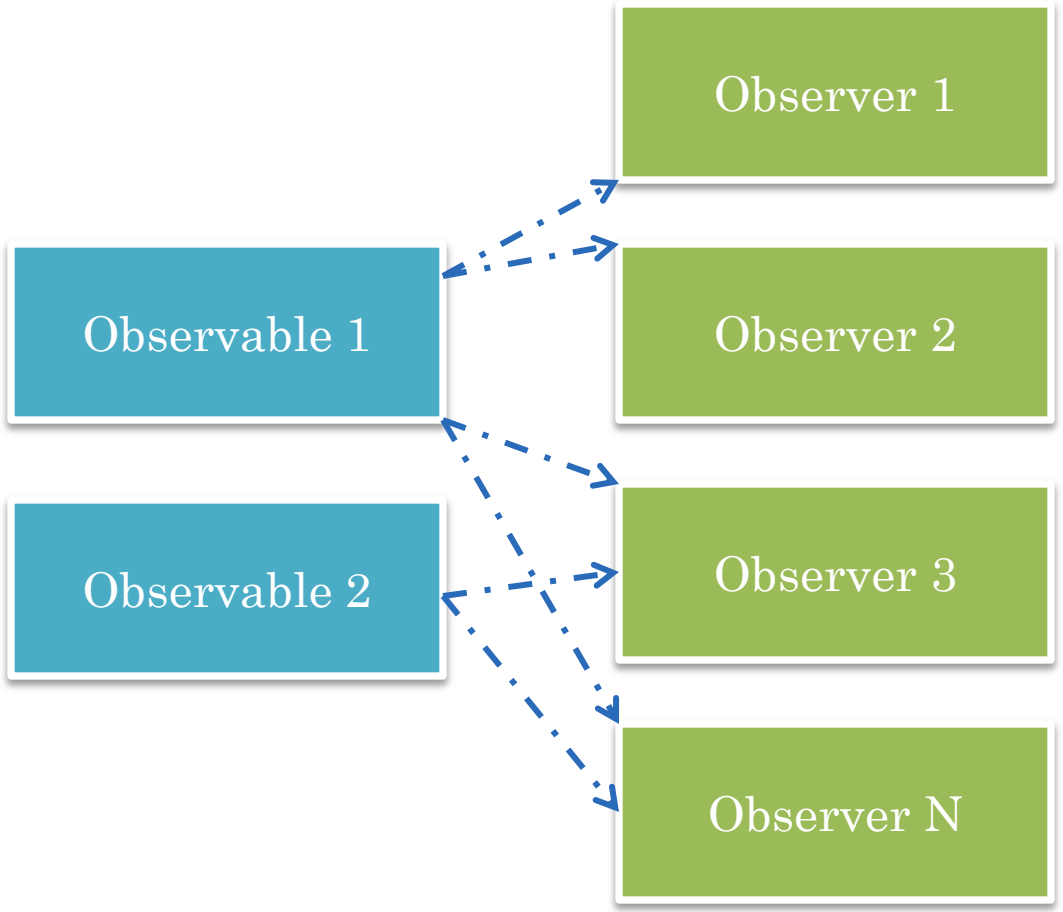
Each write method in observable calls a notification method in each observer

Notification method in observer reads model

Each student is registered with professor's listserv

When web page is updated mail sent to students

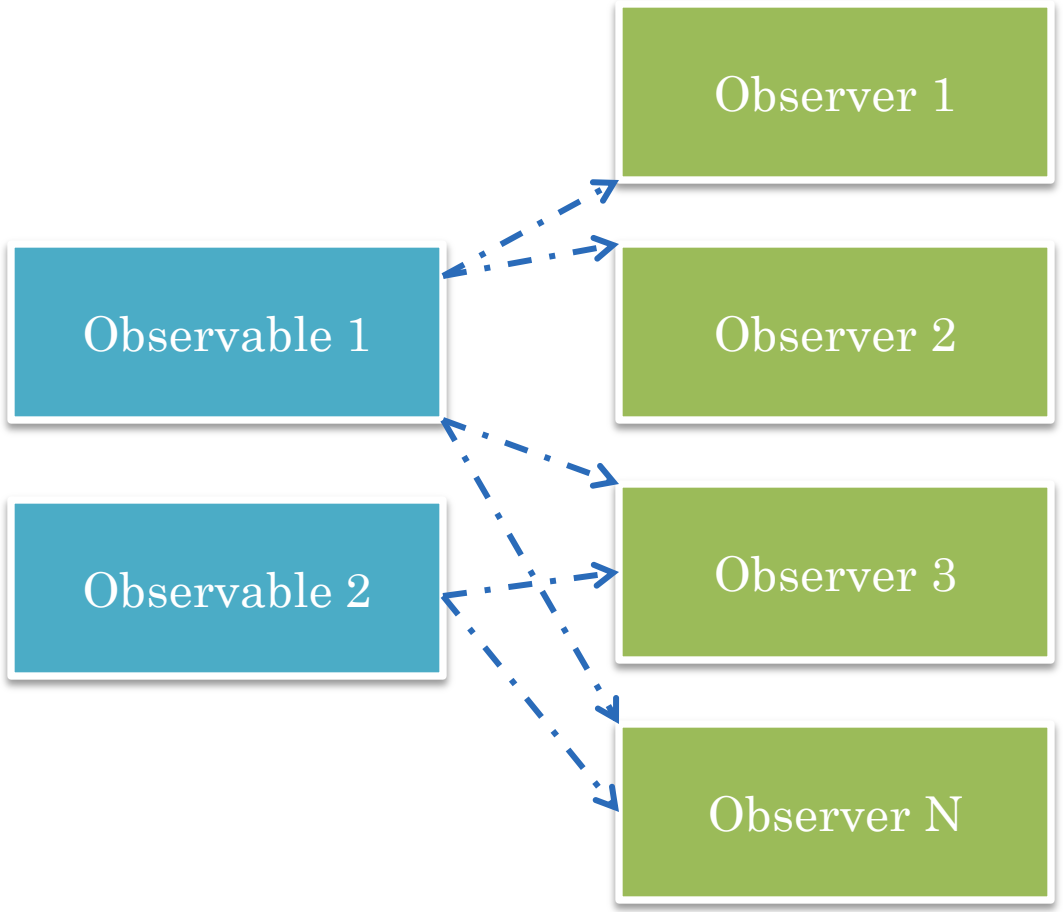
Student reads web page if mailed information is not sufficient



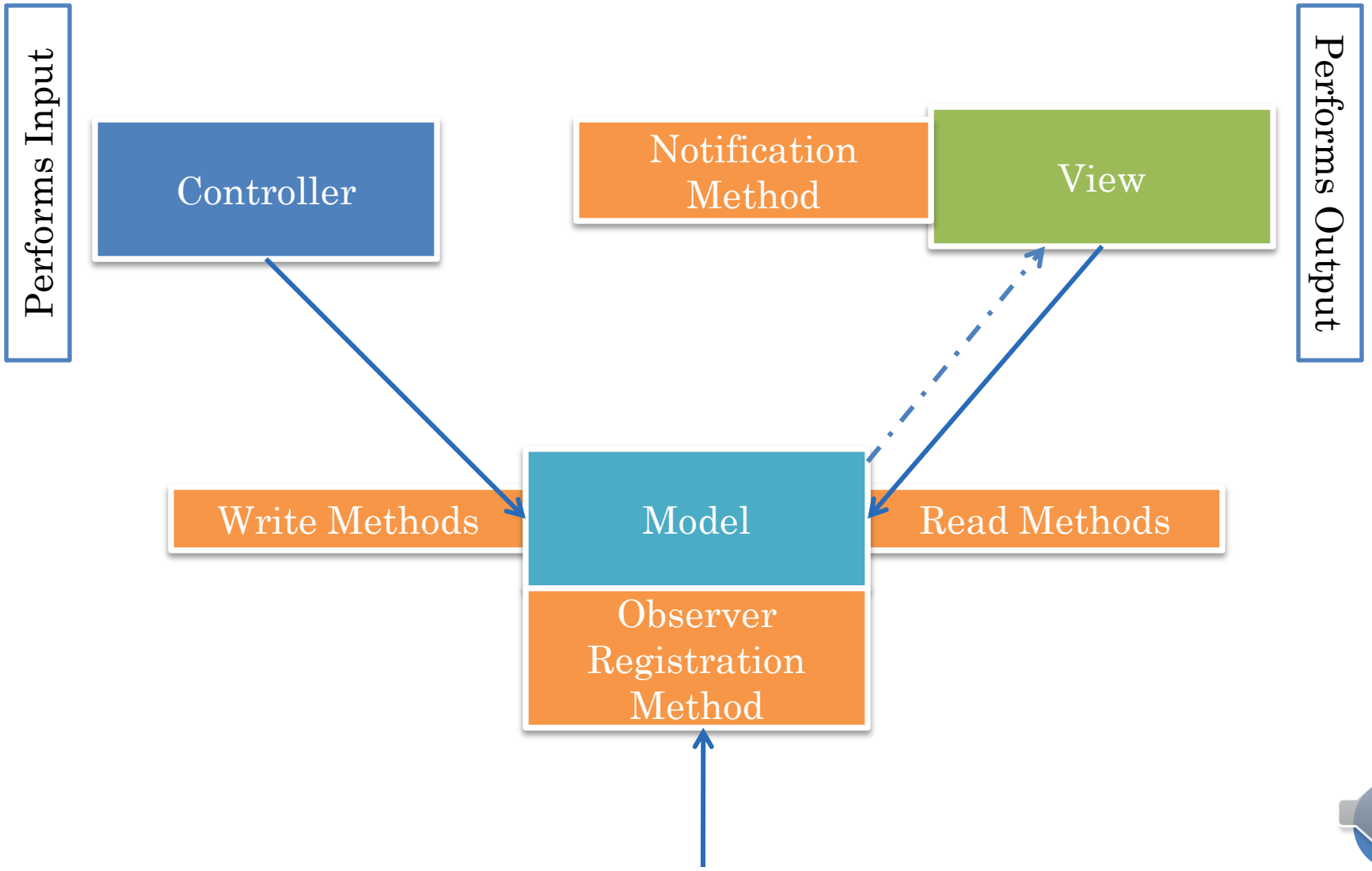
General Notification Scheme

Observers may have multiple observables with common notification method

Notification method parameter indicates which observable



Notifications in MVC Pattern



Implementation dependent issues

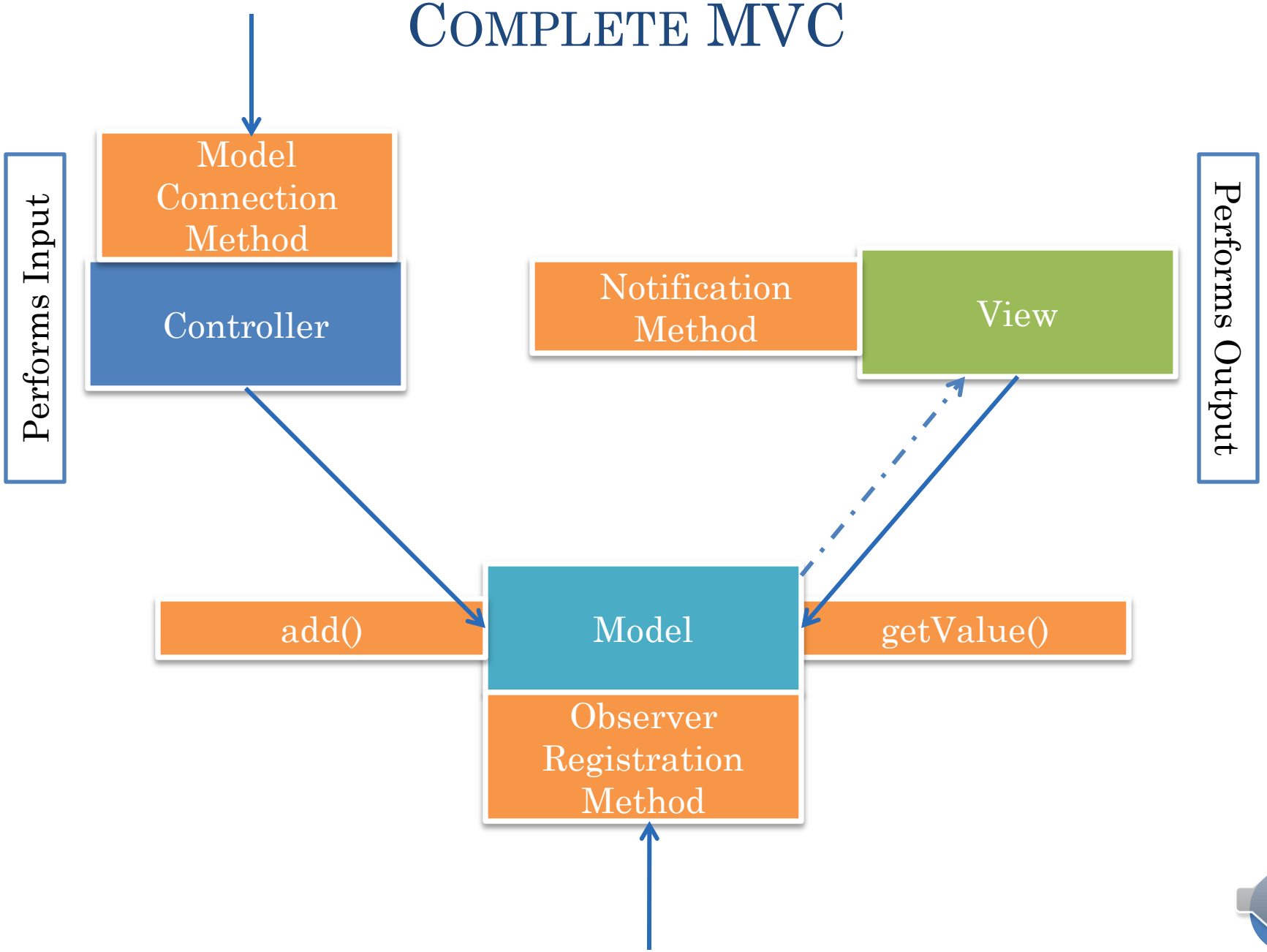
How does controller know about model?

- Model connection method invoked on it
- By model or some other program
  - Main

Who registers observer registered with observable?

- It registers itself if it knows about observable
- Model registers it if it knows about observer
- Some other code registers it
  - Main

Complete MVC



Observable and Observer

Each observer is registered with observable

Each write method in observable calls a notification method in each observer

Notification method in observer reads model

Counter Observable and Observer

Console View, JOption View

```
public interface ObservableCounter {
        public void add (int amount) ;
        public int getValue() ;
        public void addObserver(CounterObserver observer);
        public void removeObserver(CounterObserver observer);
}
```

```
public interface CounterObserver {
        public void update(ObservableCounter counter);
}
```

Called whenever model is updated

Updated model

Counter Model

```
public class AnObservableCounter implements ObservableCounter {
        int counter = 0;
        ObserverList observers = new AnObserverList();
        public void add (int amount) {
                counter += amount;
                notifyAllObservers();
        }
        public int getValue() {
                return counter;
        }
        public void addObserver(CounterObserver observer) {
                observers.addElement(observer);
                observer.update(this);
        }
        public void removeObserver(CounterObserver observer) {
                observers.removeElement(observer);
        }
        void notifyAllObservers() {
                for (int observerNum = 0; observerNum < observers.size();
                        observerNum++)
                observers.elementAt(observerNum).update(this);
        }
}
```

Give this observable initial value

Each write method notifies all!

Console View

```
public class ACounterConsoleView implements CounterObserver {
        public void update(ObservableCounter counter) {
                System.out.println("Counter: " + counter.getValue());
        }
}
```

JOption View

```
import javax.swing.JOptionPane;
public class ACounterJOptionView implements CounterObserver {
        public void update(ObservableCounter counter) {
                JOptionPane.showMessageDialog(
                        null, "Counter: " + counter.getValue());
        }
}
```

Console Controller Interface

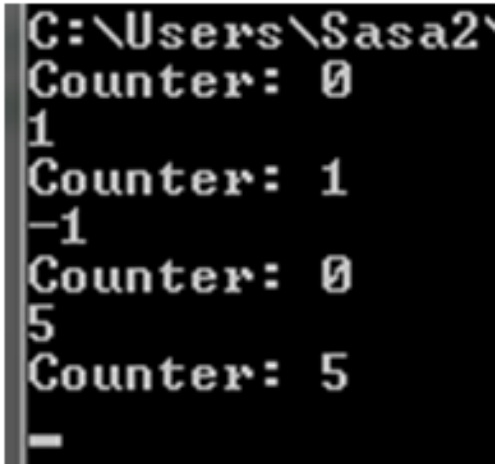
```
public interface CounterController {
        public void setModel(ObservableCounter theCounter);
        public void processInput();
}
```

Console Controller

```
public class ACounterController implements CounterController {
        ObservableCounter counter;
        public void setModel(ObservableCounter theCounter) {
                counter = theCounter;
        }
        public void processInput() {
                while (true) {
                        int nextInput = Console.readInt();
                        if (nextInput == 0) break;
                        counter.add(nextInput);
                }
        }
}
```

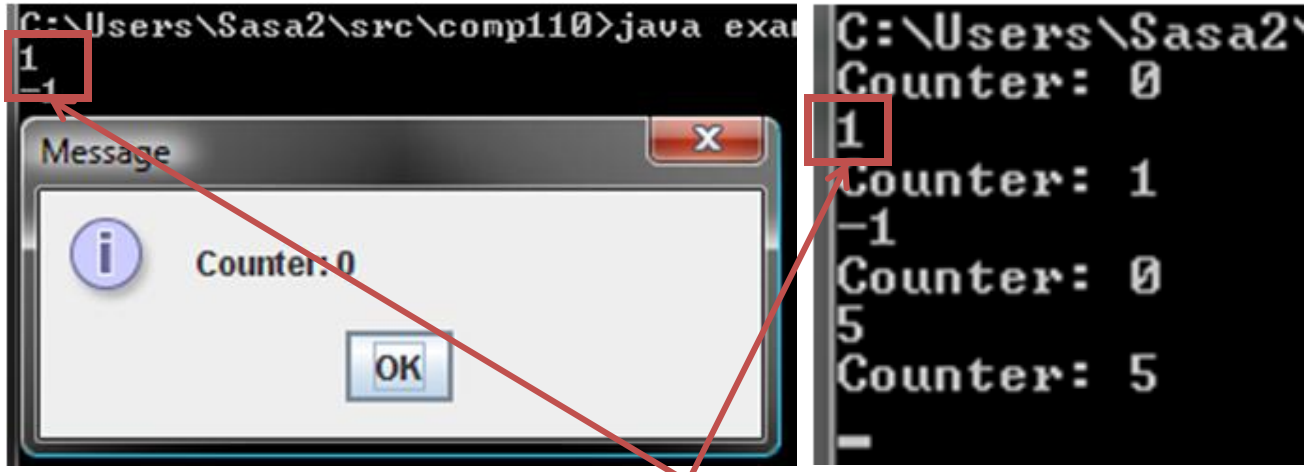
Console Main

```
public static void main (String args[]) {
        ObservableCounter model = new AnObservableCounter();
        model.addObserver(new ACounterConsoleView());
        CounterController controller = new ACounterController();
        controller.setModel(model);
        controller.processInput();
}
```



Console and JOption Main

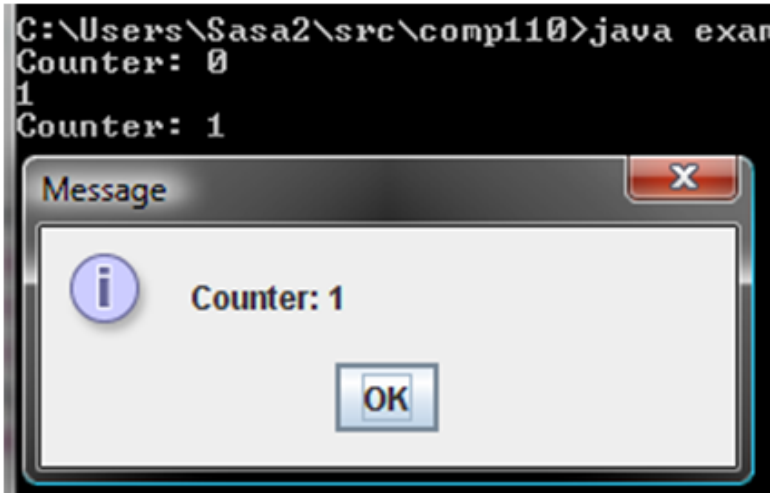
```
public static void main (String args[]) {
    ObservableCounter model = new AnObservableCounter();
    model.addObserver (new  ACounterJOptionView());
    CounterController controller = new  ACounterController();
    controller.setModel(model);
    controller.processInput();
}
```



Shared input code

Mixed UI Main

```
public static void main (String args[]) {
    ObservableCounter model = new AnObservableCounter();
    model.addObserver(new ACounterJOptionView());
    model.addObserver (new ACounterConsoleView());
    CounterController controller = new ACounterController();
    controller.setModel(model);
    controller.processInput();
}
```



Efficiency

```
public interface CounterObserver {
        public void update(ObservableCounter counter);
}
```

```
public class ACounterConsoleView implements CounterObserver {
        public void update(ObservableCounter counter) {
                System.out.println("Counter: " + counter.getValue());
        }
}
```

What if observer is in USA and observable in China?

Update must make a “long distance” call to read method (getValue()) to update counter state

Notification with Change Description

No need to call read method after notification

```
public interface CounterObserver {
        public void update(ObservableCounter counter, int newCounterVal);
}
```

```
public class ACounterConsoleView implements CounterObserver {
        public void update(ObservableCounter counter, int
newCounterVal) {
                System.out.println("Counter: " + newCounterVal));
        }
}
```

ObjectEditor Update?

```
public interface CounterObserver {
        public void update(ObservableCounter counter, int newCounterVal);
}
```

Can ObjectEditor become a view of Counter so no need to call refresh?

ObjectEditor does not know about CounterObserver and cannot implement it.

java.util.Observer and Observable

"Standard" observer interface talking arbitrary change Object argument

```
public interface java.util.Observer {
        public void update(Observable o, Object arg);
}
```

```
public class java.util.Observable {
        public void addObserver(Observer o) { … };
        public void notifyObservers() { … };
}
```

Model must be subclass of Observable

Extra

Circularity

```
public interface ObservableCounter {
        public void add (int amount) ;
        public int getValue() ;
        public void addObserver(CounterObserver observer);
        public void removeObserver(CounterObserver observer);
}
```

Cannot compile ObservableCounter without CounterObserver and vice versa

```
public interface CounterObserver {
        public void update(ObservableCounter counter);
}
```

Breaking Circularity: Multiple Stages

```
public interface ObservableCounter {
        public void add (int amount) ;
        public int getValue() ;
}
```

CounterObserver references compiled ObservableCounter

```
public interface CounterObserver {
        public void update(ObservableCounter counter);
}
```

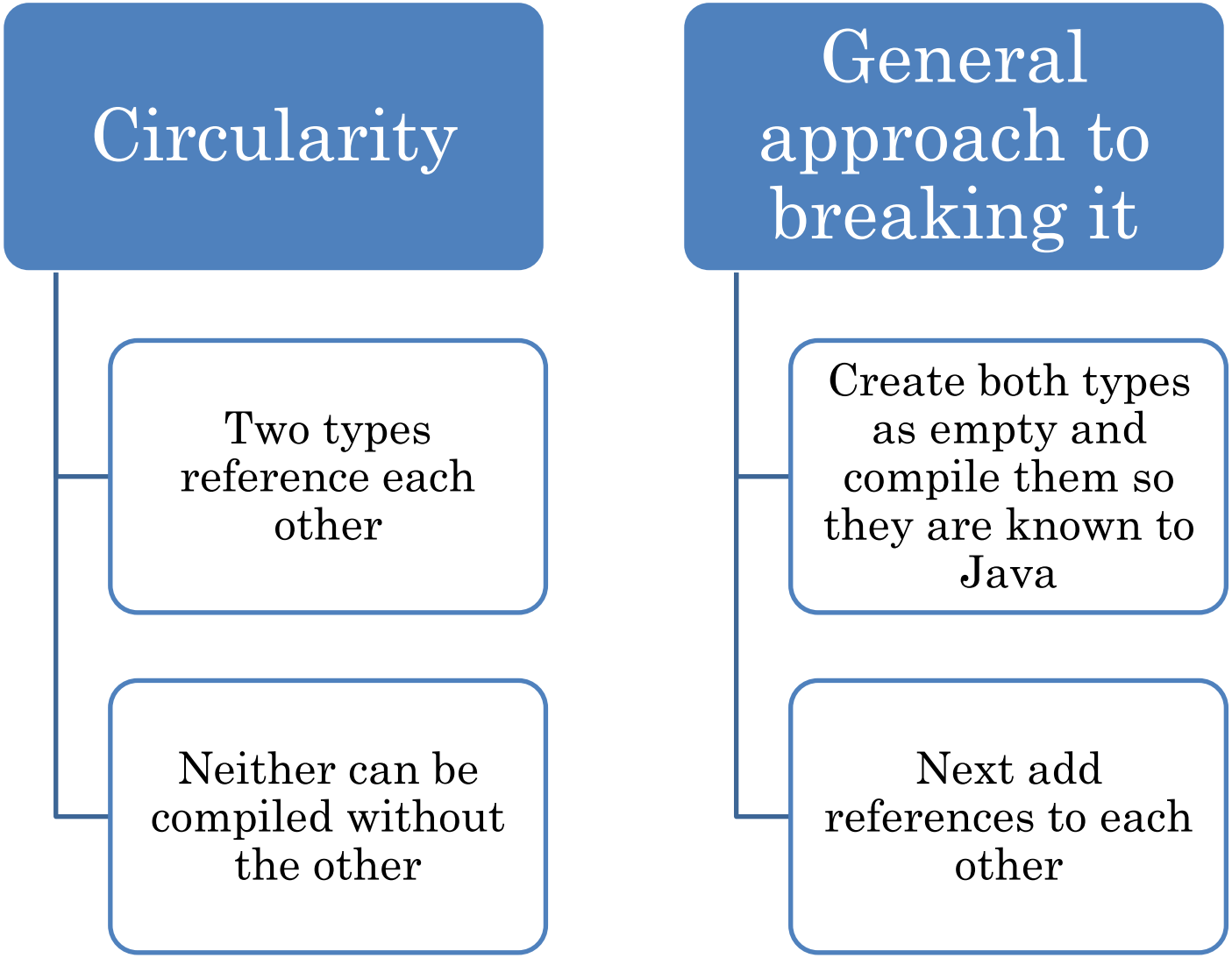
Circularity

```
public interface ObservableCounter {
        public void add (int amount) ;
        public int getValue() ;
        public void addObserver(CounterObserver observer);
        public void removeObserver(CounterObserver observer);
}
```

Recompiled observable references compiled CounterObserver

```
public interface CounterObserver {
        public void update(ObservableCounter counter);
}
```

Circularity and Breaking it



Observers that are not views

- Spreadsheet cell
  - observes cells on which it depends
- Monitoring of appliance usage
  - Each time I do setChannel() on TV event logged
- Eclipse quiz/activity plug-in
  - Observers Eclipse events
- Any big brother app!
- Counter observer?

Observers that are not views

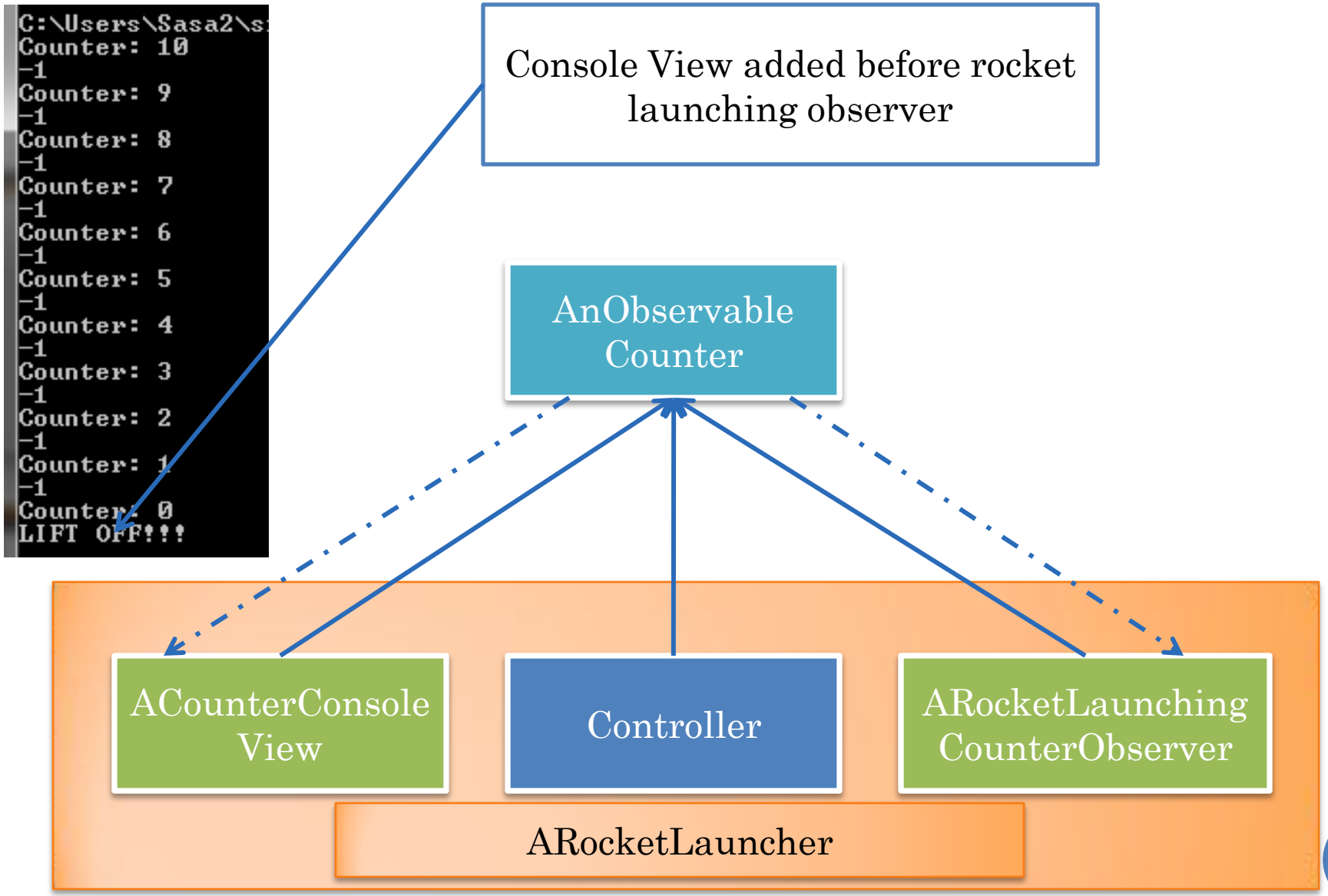
- Spreadsheet cell
  - observes cells on which it depends
- Monitoring of appliance usage
  - Each time I do setChannel() on TV event logged
- Eclipse quiz/activity plug-in
  - Observers Eclipse events
- Any big brother app!
- Counter observer?

Rocket Observer

```
C:\Users\Sasa
Counter: 10
-1
Counter: 9
-1
Counter: 8
-1
Counter: 7
-1
Counter: 6
-1
Counter: 5
-1
Counter: 4
-1
Counter: 3
-1
Counter: 2
-1
Counter: 1
-1
Counter: 0
LIFT OFF!!!
```

```
public class ARocketLaunchingCounterObserver
                implements CounterObserver {
        public void update(ObservableCounter counter) {
                if (counter.getValue() == 0)
                        launch();
        }
        private void launch() {
                System.out.println("LIFT OFF!!!");
        }
}
```

Instances created and composed

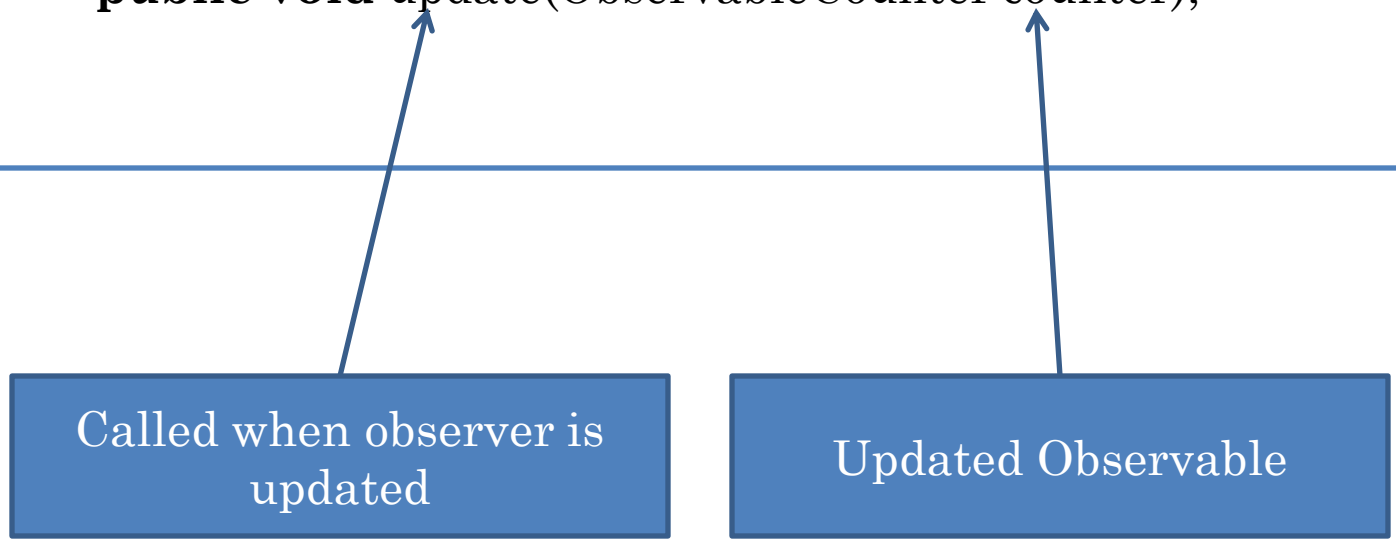


Rocket Launching Main

```
public static void main (String args[]) {
    ObservableCounter model = new AnObservableCounter();
    model.addObserver (new ACounterConsoleView());
    model.addObserver(new ARocketLaunchingCounterObserver());
    CounterController controller = new ACounterController();
    controller.setModel(model);
    controller.processInput();
}
```

Basic Notification

```
public interface CounterObserver {
        public void update(ObservableCounter counter);
}
```



Implicit Observer

Updated Observable

```
public interface CounterObserver {
        public void setObservable(ObservableCounter counter);
        public void update();
}
```

Assuming observer has only one observable

Distribution Issues

```
public interface CounterObserver {
        public void update(ObservableCounter counter);
}
```

What if observer is in USA and observable in China?

Update must make a "long distance" call to read method (getValue()) to update counter state

Notification with Change Description

No need to call read method after notification

```
package models;
public interface CounterObserver {
        public void update(ObservableCounter counter, int newCounterVal);
}
```

Java java.util.Observer

```
public interface java.util.Observer {
        public void update(Observable o, Object arg);
}
```

"Standard" observer interface talking arbitrary change Object argument

Notification with Changed Value

New value of observable attribute

```
public interface CounterObserver {
        public void update(ObservableCounter counter, int newCounterVal);
}
```

Notification with Change

Difference between new and old value of observable attribute

```
public interface CounterObserver {
        public void update(ObservableCounter counter,
                           int counterIncrement);
}
```

Observer may display change to user

Observer interested in change does not need to keep old value to determine change

Observer interested in absolute value must keep old value

Notification with New and Old Value

Old and new value of observable attribute

```
public interface CounterObserver {
        public void update (ObservableCounter counter,
                            int oldCounterValue, int newCounterValue);
}
```

Observer interested in change does not need to keep old value to determine change

Observer interested in absolute value need not keep old value

Makes observer harder to code

Notification with Single Event Object

```
public interface CounterObserver {
        public void update(
                CounterChangeEvent event);
}
```

```
public interface CounterChangeEvent {
        ObservableCounter getCounter();
        int getOldCounterValue();
        int getNewCounterValue();
}
```

- Easy to pass single object to different methods handling event
- Can make event info very elaborate
  - Time when event occurred
  - Unique ID for event
  - ….
- Callee does not have to declare parameters for event information fields not of interest
- Caller does not have to fill every value – can put null for object values such as counter and illegal values for primitives

JAVA ACTIONEVENT

```
import java.awt.Event;

public interface java.awt.ActionListener {
        public void actionPerformed(ActionEvent e);
}
```

When you edit text and hit return this event sent by JTextField, TextField widget to its listeners such as ObjectEditor

When you press a button, this event sent by Button/Jbutton to its lsisteners such as ObjectEditor

Observing Multiple Properties

```
public interface BMISpreadsheet {
        public double getHeight();
        public void setHeight(int newVal);
        public double getWeight() ;
        public void setWeight(int newWeight) ;
        public double getBMI();
}
```

Observer Inteface?

Single Coarse-Grained Update

```
public interface BMISpreadsheet {
        public double getHeight();
        public void setHeight(int newVal);
        public double getWeight() ;
        public void setWeight(int newWeight) ;
        public double getBMI();
        ….
}
```

```
public interface BMIObserver {
    public void update(
        BMISpreadsheet bmiSpreadsheet);
}
```

- Coarse grained updated
- Each setter sends the whole object
- Observer must determine which property changed

Multiple fine-grained updates

```
public interface BMISpreadsheet {
        public double getHeight();
        public void setHeight(int newVal);
        public double getWeight() ;
        public void setWeight(int newWeight) ;
        public double getBMI();
        ...
}
```

```
public interface BMIObserver {
    public void updateHeight (
        BMISpreadsheet bmi, int oldHeight, int newHeight);
    public void updateWeight(
        BMISpreadsheet bmi, int oldWeight, int newWeight);
    public void updateBMI(
        BMISpreadsheet bmi, double oldBMI, double newBMI);
}
```

SINGLE FINE-GRAINED UPDATE METHOD

```
public interface BMISpreadsheet {
        public double getHeight();
        public void setHeight(int newVal);
        public double getWeight() ;
        public void setWeight(int newWeight) ;
        public double getBMI();
        …
}
public interface BMIObserver {
    public void update(
        BMISpreadsheet bmi, String propertyName,
        Object oldValue, Object newValue);
}
```

"Wght"

"One"

New methods not needed as new properties added

Different setters calls the same update method with different types of values.

Can be used for arbitrary property values

Can make mistakes and must process property name to determine wat changed

Custom Single fine-grained update method

```
public void setHeight (int newVal) {
    int oldVal = height;
    height = newVal;
    notifyAllObservers(this, "height", oldVal, newVal);
}
```

Can make mistake

```
public void notifyAllObservers(BMISpreadsheet source, String
    propertyName, Object oldValue, Object newValue) {
    for (int index = 0; index < observers.size(); index++) {
        observers.elementAt(index).update(source, propertyName,
    oldValue, newValue);
}
```

BMIObserver