Comp 401
Practical Applications of MVC and Observer: Property Notification

Instructor: Prasun Dewan

Prerequisites

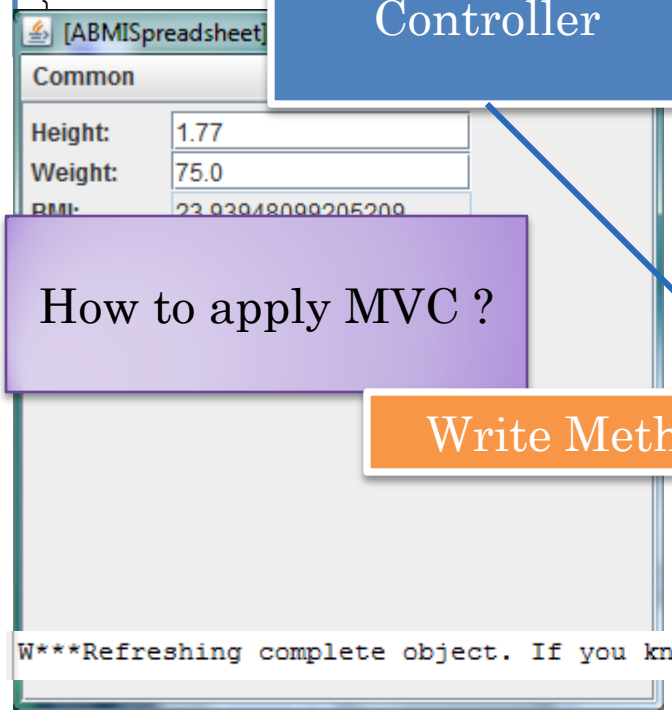
- MVC
- Composite Objects Shapes

Interface Implementation

```
public class ABMISpreadsheet implements BMISpreadsheet {
  double height;
  double weight;
  public ABMISpreadsheet(double theInitialHeight, double theInitialWeight) {
    setHeight(theInitialHeight);
    setWeight(theInitialWeight);
  }
  public double getWeight() {
    return weight;
  }
  public void setWeight(double newWeight) {
    weight = newWeight;
  }
  public double getHeight() {
    return height;
  }
  public void setHeight(double newHeight) {
    height = newHeight;
  }
  public double getBMI() {
    return weight/(height*height);
  }
}
```

Refreshing ObjectEditor from Main

```
public class ABMISpreadsheetRefreshedByMain {
  public static void main (String[] args) {
    BMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
    OEFrame oeFrame = ObjectEditor.edit(bmiSpreadsheet);
    bmiSpreadsheet.setHeight(1.77);
    bmiSpreadsheet.setWeight(75);
    oeFrame.refresh();
  }
```



Controller

Notification Method

View

How to apply MVC ?

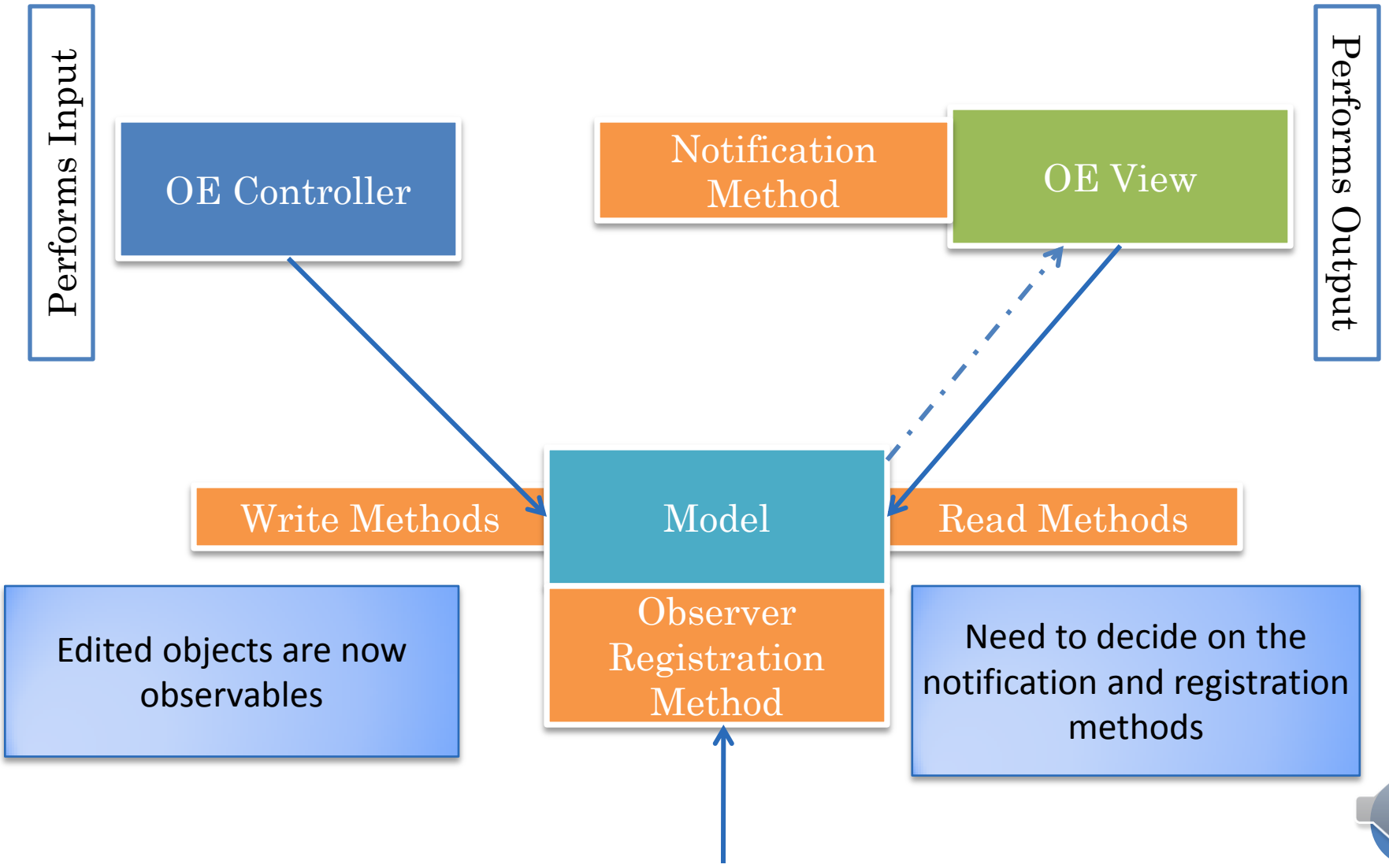
Write Methods

Model

Read Methods

Observer Registration Method

Applying Idea MVC to ObjectEditor



Adapt This How?

```
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() { … };
}
```

Must use general (de)registration method

Must use general listener type

BMISpreadsheet.java

```
public interface BMISpreadsheet {

    public double getWeight();
    public void setWeight(double newVal);

    public double getHeight();
    public void setHeight(double newVal);

    public double getBMI();

}
```

Changes?

```
public class ABMISpreadsheet implements BMISpreadsheet {
  double height;
  double weight;
  public ABMISpreadsheet(double theInitialHeight, double theInitialWeight) {
    setHeight(theInitialHeight);
    setWeight(theInitialWeight);
  }
  public double getWeight() {
    return weight;
  }
  public void setWeight(double newWeight) {
    weight = newWeight;
  }
  public double getHeight() {
    return height;
  }
  public void setHeight(double newHeight) {
    height = newHeight;
  }
  public double getBMI() {
    return weight/(height*height);
  }
}
```

Standard Single fine-grained update method

```
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import util.annotations.ObserverRegisterer;
import util.annotations.ObserverTypes;
import util.models.PropertyListenerRegisterer;
public class AnObservableBMISpreadsheet
         implements  BMISpreadsheet, PropertyListenerRegisterer {
  PropertyListenerSupport propertyListenerSupport =
         new APropertyListenerSupport();

  @ObserverRegisterer(ObserverTypes.PROPERTY_LISTENER)
  public void addPropertyChangeListener(
          PropertyChangeListener listener) {
    propertyListenerSupport.addElement(listener);
  }
  …
  public void setWeight(double newWeight) {
    double oldWeight = weight;
    double oldBMI = getBMI();
    weight = newWeight;
    if (propertyListenerSupport != null) {
      propertyListenerSupport.notifyAllListeners(
        new PropertyChangeEvent(this, "weight", oldWeight, newWeight));
      propertyListenerSupport.notifyAllListeners(
       new PropertyChangeEvent(this, "bmi", oldBMI, getBMI()));
   }
  }
```

To prevent erroneous signature

Standard registration support

Standard Bean Listener Signature

Standard event

Standard Registration Signature

```
package util.models;
import java.beans.PropertyChangeListener;
public interface PropertyListenerRegisterer {
  void addPropertyChangeListener(
           PropertyChangeListener aListener);
}
```

Support Implementation?

```
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import util.annotations.ObserverRegisterer;
import util.annotations.ObserverTypes;
import util.models.PropertyListenerRegisterer;
public class AnObservableBMISpreadsheet
         implements  BMISpreadsheet, PropertyListenerRegisterer {
  PropertyListenerSupport propertyListenerSupport =
         new APropertyListenerSupport();

  @ObserverRegisterer(ObserverTypes.PROPERTY_LISTENER)
  public void addPropertyChangeListener(
          PropertyChangeListener listener) {
    propertyListenerSupport.addElement(listener
  }
  …
  public void setWeight(double newWeight) {
    double oldWeight = weight;
    double oldBMI = getBMI();
    weight = newWeight;
    if (propertyListenerSupport != null) {
      propertyListenerSupport.notifyAllListeners(
        new PropertyChangeEvent(this, "weight", oldWeight, newWeight));
      propertyListenerSupport.notifyAllListeners(
       new PropertyChangeEvent(this, "bmi", oldBMI, getBMI()));
   }
  }
```

Standard registration support

No need to subclass to get automatic notification as in Observable

History + Notify All

```
public class APropertyListenerSupport implements PropertyListenerSupport {
  public final int MAX_SIZE = 50;
  PropertyChangeListener[] contents = new PropertyChangeListener[MAX_SIZE];
  int size = 0;
  public int size() {
    return size;
  }
  public PropertyChangeListener elementAt (int index) {
    return contents[index];
  }
  boolean isFull() {
    return size == MAX_SIZE;
  }
  public void addElement(PropertyChangeListener l) {
    if (isFull())
      System.out.println("Adding item to a full history");
   else {
     contents[size] = l;
     size++;
   }
}
  public void notifyAllListeners(PropertyChangeEvent event) {
    for (int index = 0; index < size(); index++) {
      elementAt(index).propertyChange(event);
    }
  }
}
```

Modelled after standard Java PropertyChangeSupport

Standard Listener and Event

```
package java.beans;
public interface PropertyChangeListener {
     public void propertyChange (PropertyChangeEvent evt);
}
```

```
package java.beans;
public class  PropertyChangeEvent extends java.util.EventObject {
    public PropertyChangeEvent (Object source, String propertyName,
                                Object oldValue, Object newValue) {…}
    public Object getNewValue() {…}
    public Object getOldValue() {…}
    public String getPropertyName() {…}
    ….
}
```

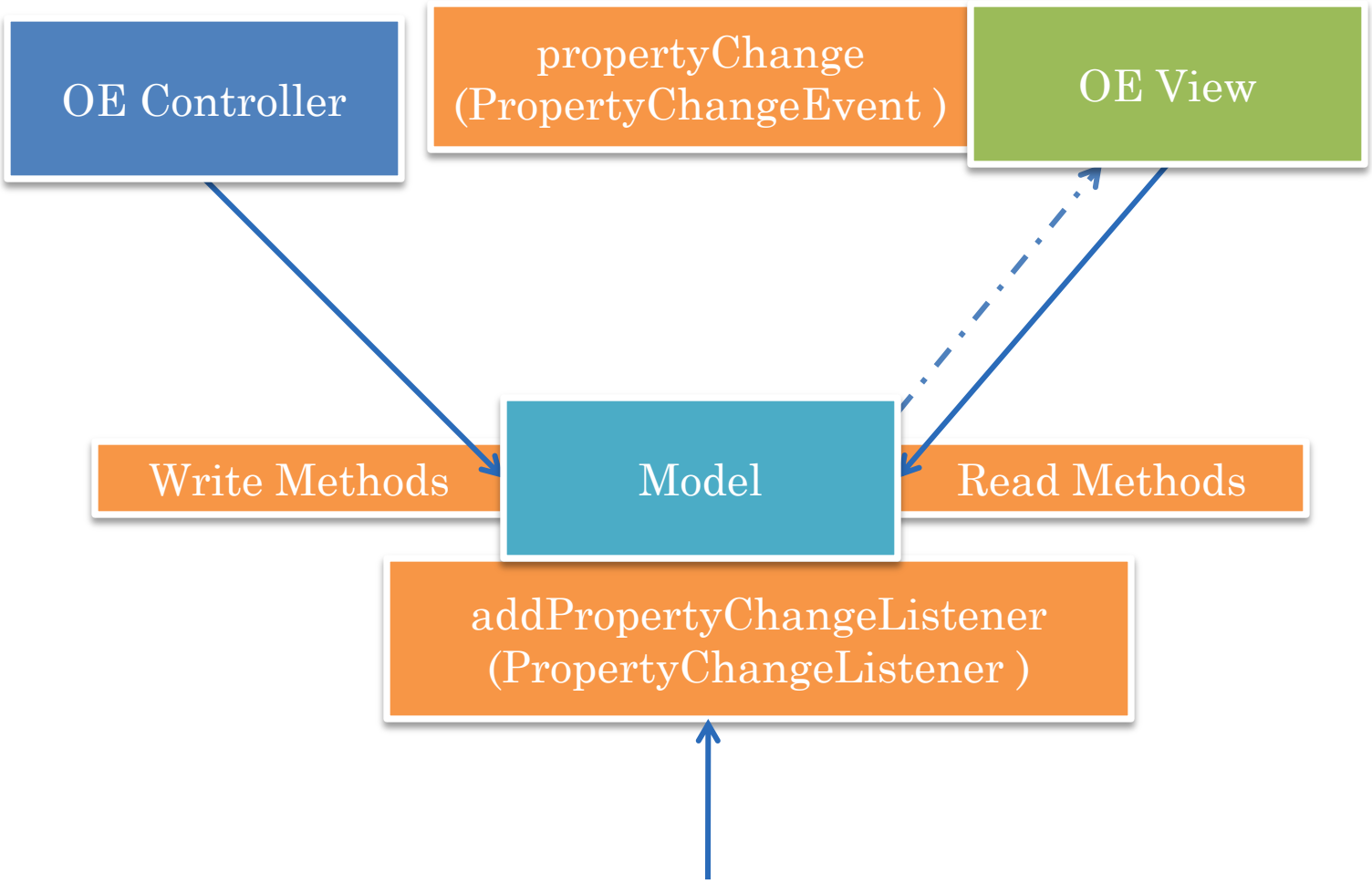
Needed only if you are defining your own view/observable

ObjectEditor Observer Protocol

- ObjectEditor implements the standard PropertyChangeListener interface

```
public class ObjectEditorView implements PropertyChangeListener  {
    public void propertyChange(PropertyChangeEvent event) {
            …
    }
}
```

Communicating Bean Events



ObjectEditor Observer Protocol

- If the class of a displayed object defines the standard method:
- @util.annotations.ObserverRegisterer(util.annotations.ObserverTypes.*PROPERTY_LISTENER)*
  - **public void** addPropertyChangeListener (PropertyChangeListener l)
  - Annotation or util.models.PropertyListenerRegisterer interface to ensure method signature is right
- ObjectEditor calls the method to register itself as an observer
- Method should store a reference to ObjectEditor and other observers
- Cannot define own PropertyChangeListener with same name and methods as predefined, ObjectEditor expects predefined interface

ObjectEditor Observer Protocol

- A property changing method can now call the propertyChange(PropertyChangeEvent arg) defined by PropertyChangeListener to inform ObjectEditor and other observers about change

```
public void setWeight(double newWeight) {
    double oldWeight = weight;
    double oldBMI = getBMI();
    weight = newWeight;
    propertyListenerSupport.notifyAllListeners( new
      PropertyChangeEvent(this, "weight", oldWeight, newWeight));
    propertyListenerSupport.notifyAllListeners(new
      PropertyChangeEvent(this, "bmi", oldBMI, getBMI()));
  }
```

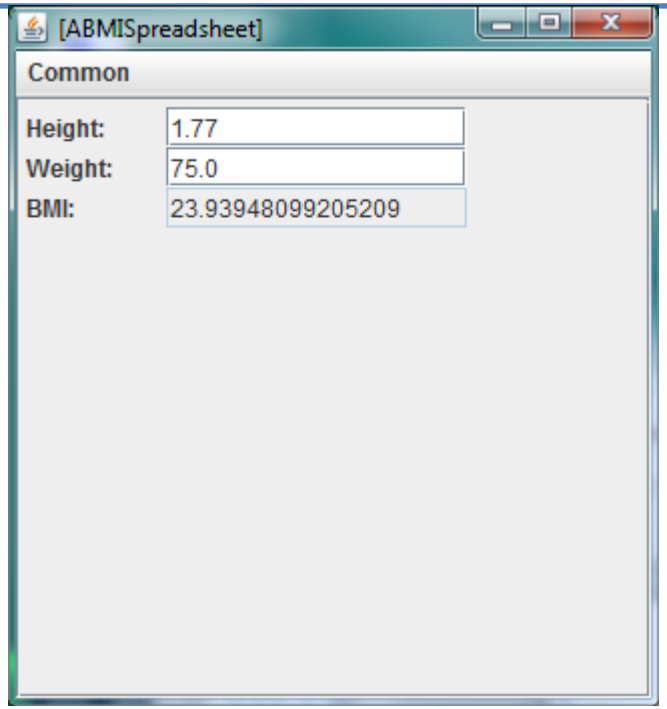
Object Editor View

- The implementation of this method in ObjectEditor updates the display

```
public class ObjectEditorView implements
        java.beans.PropertyChangeListener  {
    public void propertyChange (PropertyChangeEvent arg) {
        // update display of property arg.getPropertyName()
        // to show arg.getNewValue()
        …
    }
}
```

Refreshing ObjectEditor from Main

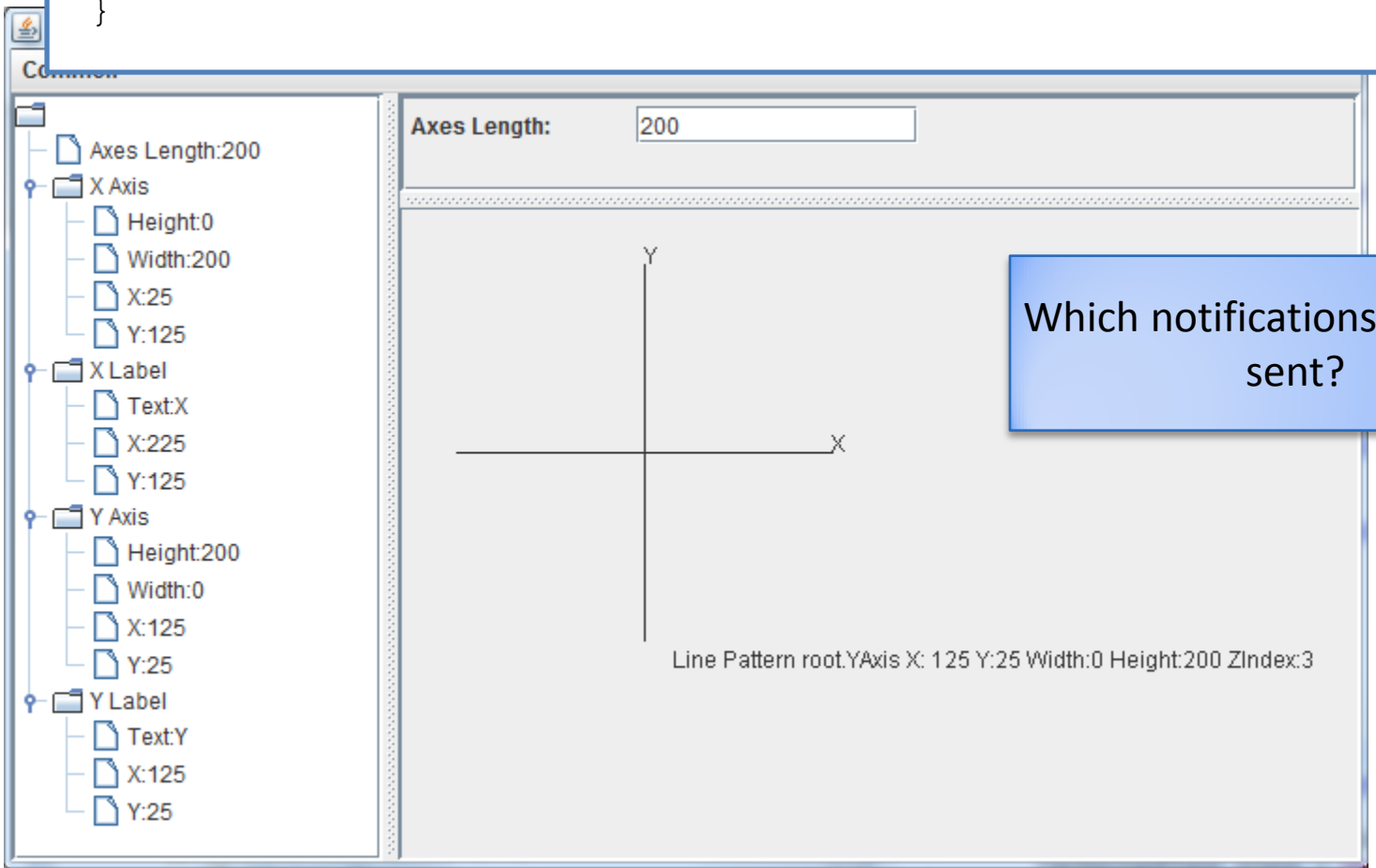
```
public class AnObservableBMISpreadsheetDemoer {
  public static void main (String[] args) {
    BMISpreadsheet bmiSpreadsheet =
                          new AnObservableBMISpreadsheet();
    ObjectEditor.edit(bmiSpreadsheet);
    bmiSpreadsheet.setHeight(1.77);
    bmiSpreadsheet.setWeight(75);
  }
}
```



No warning now

Composite Objects?

```
public static void main (String[] args) {
    CartesianPlane cartesianPlane =
      new AnObservableCartesianPlane(200, 125, 125);
    ObjectEditor.edit(cartesianPlane);
    cartesianPlane.setAxesLength(100);
 }
```



Which notifications should be sent?

Unobservable Cartesian Plane

Need notification

```
public void setAxesLength(int anAxesLength) {
    axesLength = anAxesLength;
    xAxis.setWidth(axesLength);
    yAxis.setHeight(axesLength);
    xAxis.setX(toXAxisX());
    xAxis.setY(toXAxisY());
    yAxis.setX(toYAxisX());
    yAxis.setY(toYAxisY());
    xLabel.setX(toXLabelX());
    xLabel.setY(toXLabelY());
    yLabel.setX(toYLabelX());
    yLabel.setY(toYLabelY());
}
```

Notification only for atomic properties

No notifications for composite properties/elements unless new object assigned to them

New objects should be assigned if necessary

Observable Cartesian Plane

```
public class AnObservableCartesianPlane extends ACartesianPlane
                                  implements ObservableCartesianPlane {
  PropertyListenerSupport propertySupport =
                new APropertyListenerSupport();
  public AnObservableCartesianPlane (int theAxesLength,
                                  int theOriginX, int theOriginY ) {
    super(theAxesLength, theOriginX, theOriginY);
    xAxis = new AnObservableLine(toXAxisX(), toXAxisY(), axesLength, 0);
    yAxis = new AnObservableLine(toYAxisX(), toYAxisY(), 0, axesLength);
    xLabel = new AnObservableStringShape ("X", toXLabelX(), toXLabelY());
    yLabel = new AnObservableStringShape ("Y", toYLabelX(), toYLabelY());
 }
  public void setAxesLength(int newVal) {
    int oldVal = getAxesLength();
    super.setAxesLength(newVal);
    propertySupport.notifyAllListeners(
       new PropertyChangeEvent(this, "axesLength",
                                  oldVal, newVal));
  }
  …
}
```

No notifications for composite properties/elements unless new object assigned to them

Atomic Objects Announcing Event

AStringShape

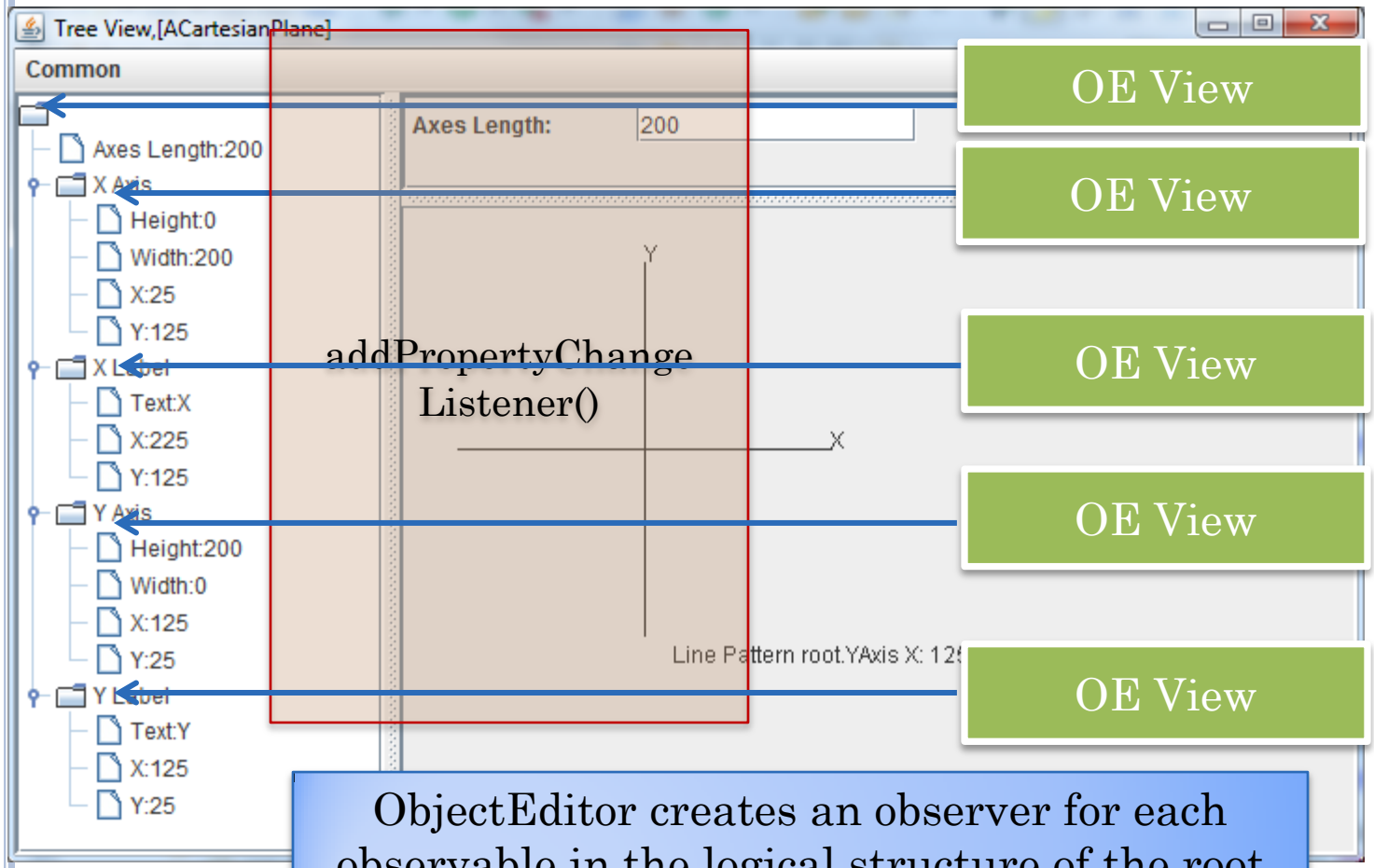
```
public void setX(int newX) {
  x = newX;
}
```

If a single setX (anX) shared by all classes in a project, then a single X setter method must be changed.

AnObservableStringShape

```
public void setX(int newVal) {
  int oldVal = getX();
  super.setX(newVal);
  propertySupport.notifyAllListeners(
    new PropertyChangeEvent(this, "X", oldVal, newVal));
}
```

Targets of Registration Method



ObjectEditor creates an observer for each observable in the logical structure of the root object displayed to ObjectEditor.edit()

Refreshing Objects with Object Properties

- Usually only primitive values refreshed
  - Unless object properties/elements (must be) assigned new values
  - Can result in multiple events being sent for the same high level action (resize axes in Cartesian Plane)
  - If a composite value is announced ObjectEditor has to create a new set of views for the composite value which is likely to be more expensive than processing multiple notifications
- An ObjectEditor view registers itself as an observer of each observable in the logical structure of the object passed to ObjectEditor.edit()

Alternative Coarse-Grained Observable

```
public class AnInefficientObservableCartesianPlane
    extends ACartesianPlane implements Obse
  PropertyListenerSupport propertySupport
              new APropertyListenerSupport(
  public AnInefficientObservableCartesi
              int theAxesLength, int theOriginX, int theOriginY ) {
    super(theAxesLength, theOriginX, theOriginY);
  }
 public void setAxesLength(int newVal) {
    int oldVal = getAxesLength();
    super.setAxesLength(newVal);
    propertySupport.notifyAllListeners(new
"axesLength", oldVal,
newVal));
    propertySupport. notifyAllListeners(new PropertyChangeEvent(this,
"XAxis", xAxis, xAxis));
    propertySupport. notifyAllListeners(ne
"YAxis", yAxis, yAxis));
    propertySupport.notifyAllListeners(new
"XLabel", xLabel, xLabel));
    propertySupport.notifyAllListeners(new PropertyChangeEvent(
   this, "YLabel", yLabel, yLabel));
    }}
```

Ask ObjectEditor to recreate and redraw the axes and label!

Inefficient, but will work

All getters of label and lines will be called as view does not know which part of label/line changes.

What if we announce some but not all changes

Partially Observable Object

- An object may announce changes to only some of its properties
  - A label may announce location change but not size changes.
- Only some of the values assigned to properties of an object may themselves announce changes
  - Labels of ACartesianPoint may announce changes but not its Axes.
- For full autorefresh
  - Every change in logical structure must be announced
  - Tedious
  - That is why ObjectEditor calls refresh after each method call

Dependent Values in Children

```
public void setAxesLength(int anAxesLength) {
    axesLength = anAxesLength;
    /*
    xAxis.setWidth(axesLength);
    yAxis.setHeight(axesLength);
    xAxis.setX(toXAxisX());
    xAxis.setY(toXAxisY());
    yAxis.setX(toYAxisX());
    yAxis.setY(toYAxisY());
    xLabel.setX(toXLabelX());
    xLabel.setY(toXLabelY());
    yLabel.setX(toYLabelX());
    yLabel.setY(toYLabelY());
    */
}
```

What if component objects have readonly properties dependent on parent properties?

Label and line have only getters defined in terms of parent axesLength and other variables.

Announcing Children Dependent Values

```
public void setAxesLength(int anAxesLength) {
  axesLength = anAxesLength;
  …
  propertySupport. notifyAllListeners(
     new PropertyChangeEvent(xLabel, "X", oldXLabelX, xLabelX));
  propertySupport. notifyAllListeners(
     new PropertyChangeEvent(xLabel, "Y", oldXLabelY, xLabelY));
  …
}
```

Parent write method can announce changes to children properties

Making an Object Auto Refreshable

- Do not call refresh() method of ObjectEditor, instead make all atomic shapes observable
- For each class C with one or more atomic properties do the following.
- Implement the util.models.PropertyListenerRegisterer interface
  - This means you have to implement the addPropertyChangeListener (java.beans.PropertyChangeListener l)
  - Delcare an instance variable of type PropertyListenerSupport holding an instance of APropertyListenerSupport
  - addPropertyChangeListener will simply ask the PropertyListenerSupport instance to add the listener to its collection
- Announce property change events in the setters of atomic properties
  - Create an instance of *Java* java.beans.PropertyChangeEvent to describe the change, giving it this(the object whose property was changed), propertyName, old property value, new property value
    - Do not define your own PropertyChangeEvent with the same naming conventions!
  - Ask the instance of PropertyListenerSupport to broadcast this event to all of the listeners it keeps track of
- Make sure a composite object does not create a new object in a getter for an object property.

Change not Reflected in an Observer

- The class of the observable does not define a registration method.
- The observer does not call this method in the observer or calls it in is parent or child
- The observer is not sent changes to the component.
- The observer does not correctly react to the change.
- Use print statements or breakpoints to debug?

ObjectEditor vs. Your Own Observer

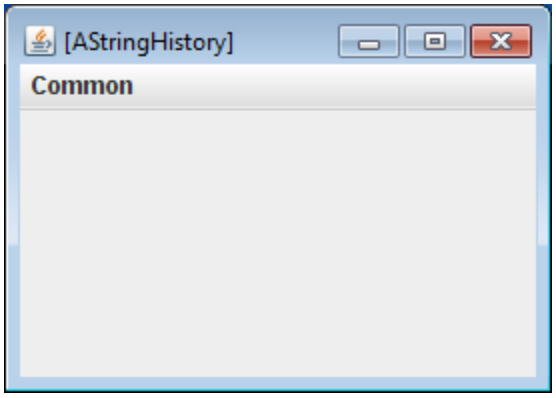
- ObjectEditor automatically registers with observable based on name and argument of registration method
  - The argument must be predefined PropertyChangeListener
- You must write your own code to register observer

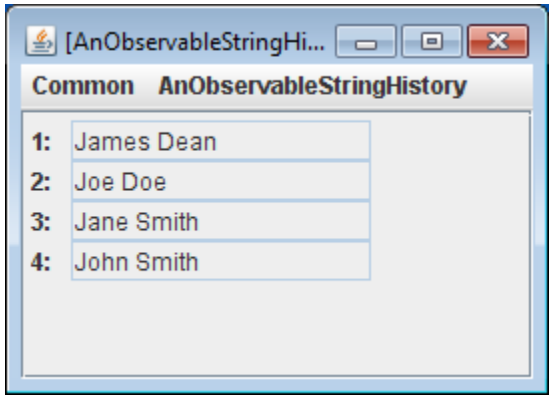
Common Situation in Partially Observable Object

- "Some value is refreshed correctly in the main window but not in the graphics window."
- An object/property is either displayed in the graphics window or in the main window - not both. Point's x and y coordinates not displayed as text fields in the main window.
- Real problem: you have two copies of the same value, one displayed in the main window and one in the graphics window, and you have announced changes to the former and not the latter.

Collection Notifications?

```
public static void createAndDisplayHistory(
    StringHistory stringHistory) {
    ObjectEditor.edit(stringHistory);
    ThreadSupport.sleep (1000);
    stringHistory.addElement("James Dean");
    stringHistory.addElement("Joe Doe");
    stringHistory.addElement("Jane Smith");
    stringHistory.addElement("John Smith");
}
```





How to make it Observable?

```
public class AStringHistory implements StringHistory {
        public final int MAX_SIZE = 50;
        String[] contents = new String[MAX_SIZE];
        int size = 0;
        public int size() { return size;}
        public String elementAt (int index) {  return contents[index]; }
        boolean isFull() { return size == MAX_SIZE; }
        public void addElement(String element) {
           if (isFull())
                   System.out.println("Adding item to a full history");
           else {
                   contents[size] = element;
                   size++;
           }
    }
}
```

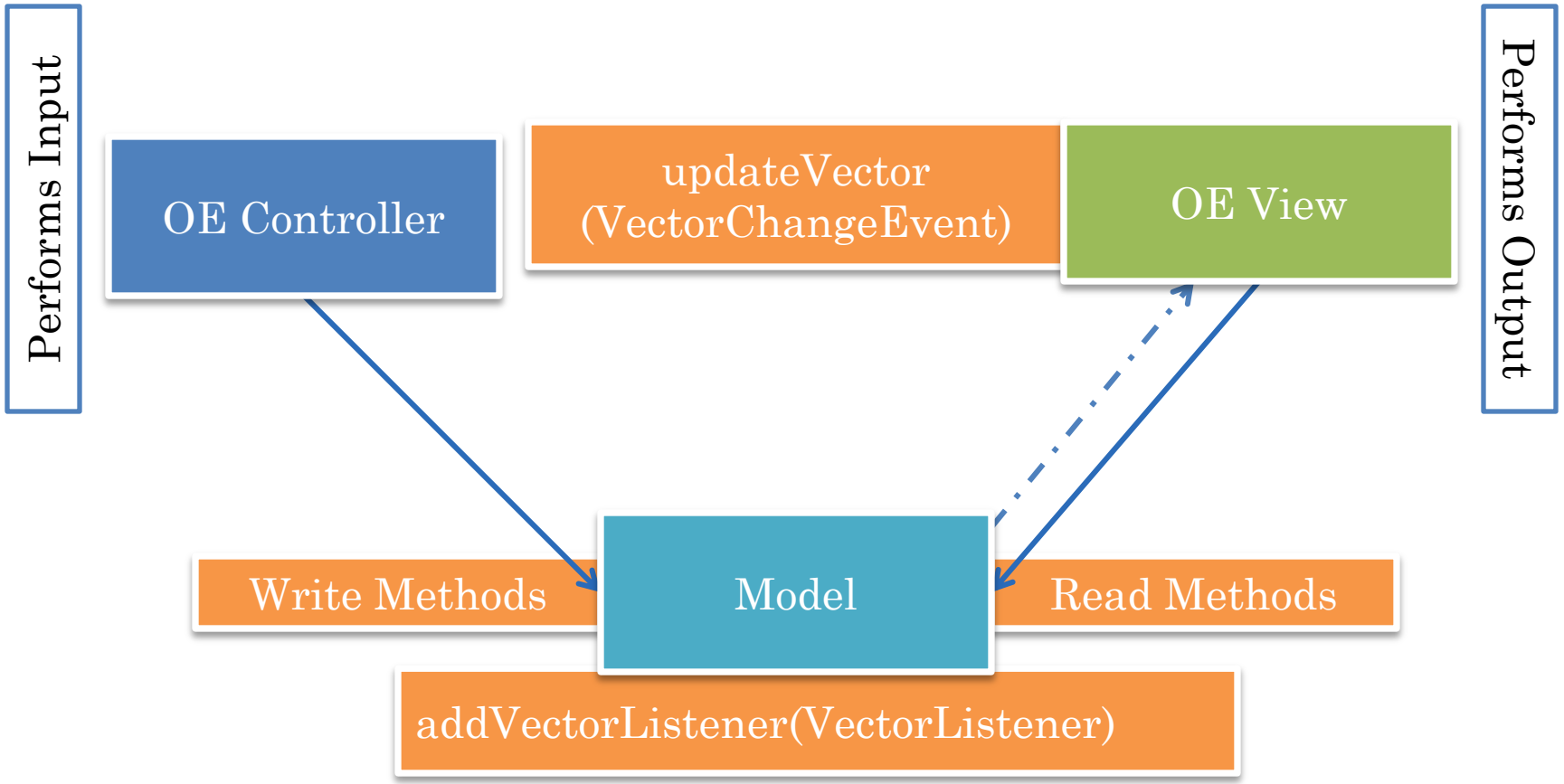
Observable String History

```
import util.models.VectorChangeEvent;
import util.models.VectorListener;
import util.models.VectorListenerRegisterer;
public class AnObservableStringHistory
         implements StringHistory,  VectorListenerRegisterer{
  VectorListenerSupport vectorListenerSupport =
    new AVectorListenerSupport();
  …
  public void addElement(String element) {
    if (isFull())
      System.out.println("Adding item to a full history");
   else {
     contents[size] = element;
     size++;
     vectorListenerSupport.notifyAllListeners(
       new VectorChangeEvent(this, VectorChangeEvent.AddComponentEvent,
             size -1, null, element, size));
   }
  }
  @ObserverRegisterer(ObserverTypes.VECTOR_LISTENER)
  public void addVectorListener(VectorListener aListener) {
    vectorListenerSupport.addElement(aListener);
  }
  …
```

Vector Change Support

```
public class AVectorListenerSupport implements VectorListenerSupport {
public final int MAX_SIZE = 50;
  VectorListener[] contents = new VectorListener[MAX_SIZE];
  int size = 0;
  public int size() {
    return size;
  }
  public VectorListener elementAt (int index) {
    return contents[index];
  }
  boolean isFull() {
    return size == MAX_SIZE;
  }
  public void addElement(VectorListener l) {
   if (isFull())
      System.out.println("Adding item to a full history");
   else {
     contents[size] = l;
    size++;
   }
  }
  public void notifyAllListeners(VectorChangeEvent event) {
    for (int index = 0; index < size(); index++) {
      elementAt(index).updateVector(event);
    }
  }
}
```

Collection Notifications



Works for any variable sized collection defined using ObjectEditor convention (originally derived for Vectors)

VectorListener and VectorListenerRegisterer Interfaces

```
package util.models;
public interface VectorListener {
   public void updateVector(VectorChangeEvent evt);
}
```

```
package util.models;
public interface VectorListenerRegisterer {
   public void addVectorListener(
                        VectorListener aListener);
}
```

VectorChangeEvent

```
package util.models;
public class VectorChangeEvent {
 Object source;
 Object otherSource;
  // constants for event types
   public static final int AddComponentEvent = 2
                           DeleteComponentEvent = 2,
                           ChangeComponentEvent = 3,
                           InsertComponentEvent = 4,
                           CompletedComponentsEvent = 5,
                           ClearEvent = 6,
                           UndefEvent = 1000;
  // constructor, oldObject can be null when no value  is replaced
  public VectorChangeEvent(Object theSource, int type, int posn,
       Object oldObject, Object newObject, int newSize) {..}
```

Collection Notifications

```
import util.models.VectorChangeEvent;
import util.models.VectorListener;
import util.models.VectorListenerRegisterer;
public class AnObservableStringHistory
         implements StringHistory,  VectorListenerRegisterer{
  VectorListenerSupport vectorListenerSupport =
    new AVectorListenerSupport();
  …
  public void addElement(String element) {
    if (isFull())
      System.out.println("Adding item to a full history");
   else {
     contents[size] = element;
     size++;
     vectorListenerSupport.notifyAllListeners(
       new VectorChangeEvent(this, VectorChangeEvent.AddComponentEvent,
             size -1, null, element, size));
   }
  }
  @ObserverRegisterer(ObserverTypes.VECTOR_LISTENER)
  public void addVectorListener(VectorListener aListener) {
```

```
public VectorChangeEvent(Object theSource, int type,
   int posn, Object oldObject, Object newObject,
   int newSize)
```

TABLE?

```
// associates key with value, returning last value associated with key
public <ValueType> put (<KeyType> key, <ValueType> value);
// returns last value associated with key, or null if no association
public <ValueType> get (<KeyType> key);
// optional, removes associated value, and returns it or null
public <ValueType> remove(<KeyType> key);
```

Necessary but not sufficient to displays all keys and elements

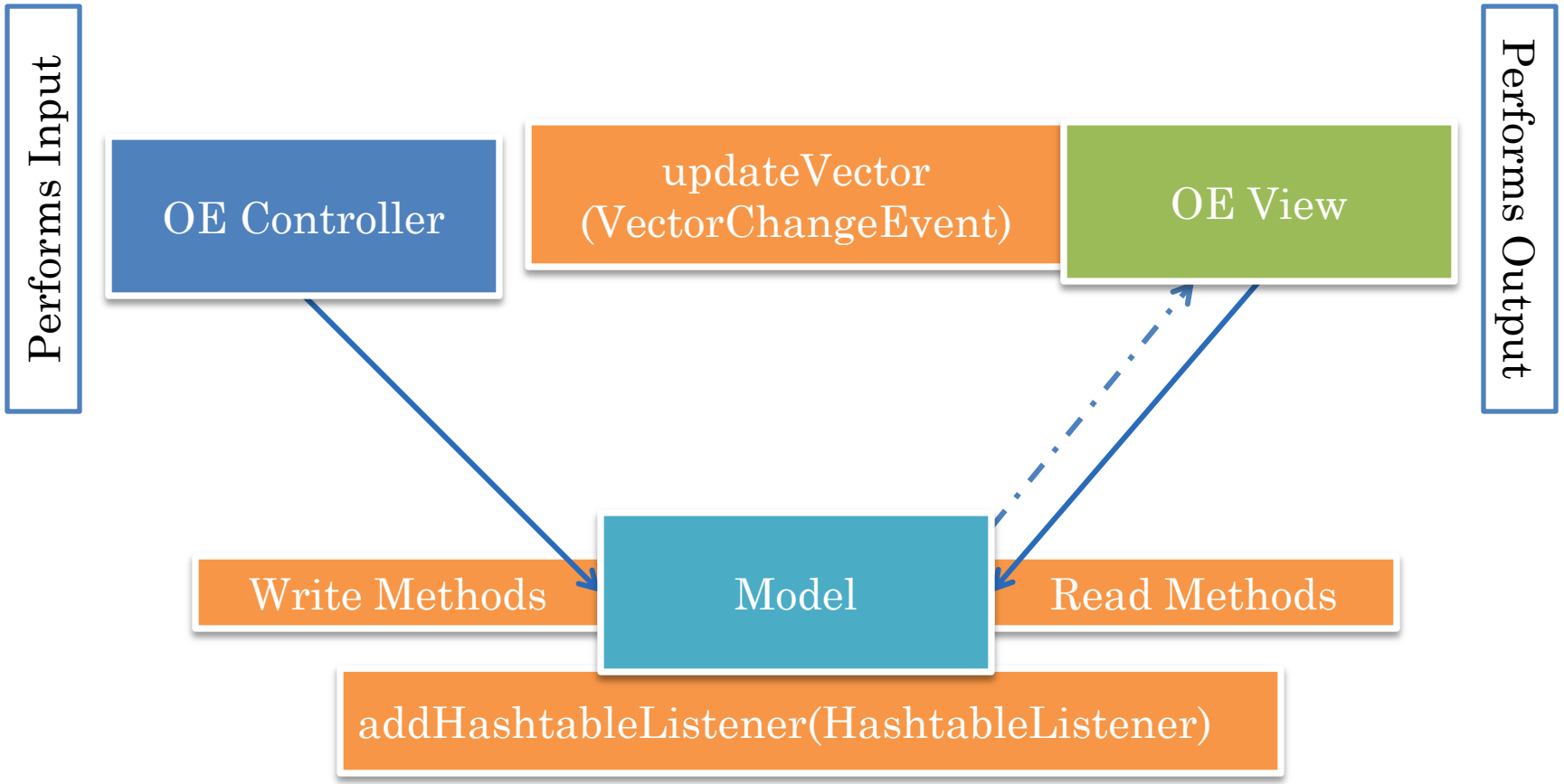
OE does not know the keys

Object can announce each put (and optional remove)

Table Listener

```
package util.models;
public interface HashtableListener {
  public void keyPut(Object source, Object key,
                     Object value,  int newSize);
  public void keyRemoved(Object source, Object key, int newSize);
}
```

Table Notifications



Works for any variable sized collection defined using ObjectEditor convention (originally derived for Vectors)

Observers

- Different forms of observers
- Some differences in syntax
  - Using term "Listener" instead of "Observer"
- Some differences in amount of information conveyed to observer about change
  - Send whole object
  - Java Observer gets whole object, PropertyChangeListener gets property changed
- Different kinds of listeners for properties and list components
  - Property changed vs. element inserted, deleted, replaced
- Some difference in generality of notification scheme
  - update(ObservableCounter) cannot be invoked on ObjectEditor
  - ObjectEditor cannot implement an observer interface defined by us.