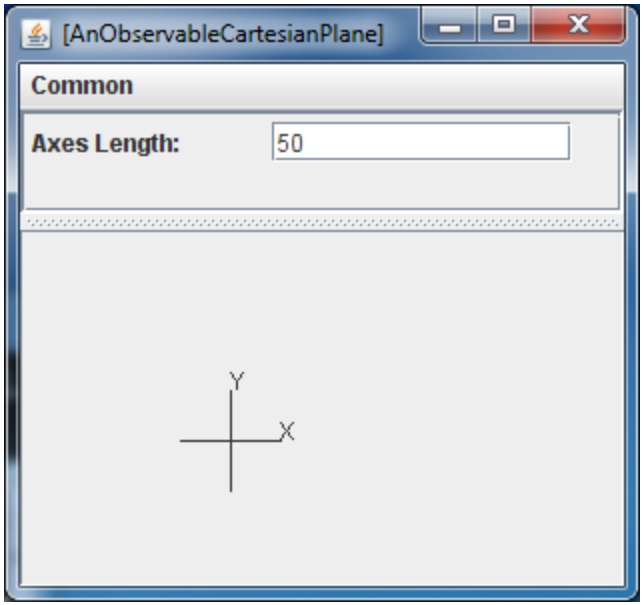
Comp 401
MVC and Graphics/Window Systems

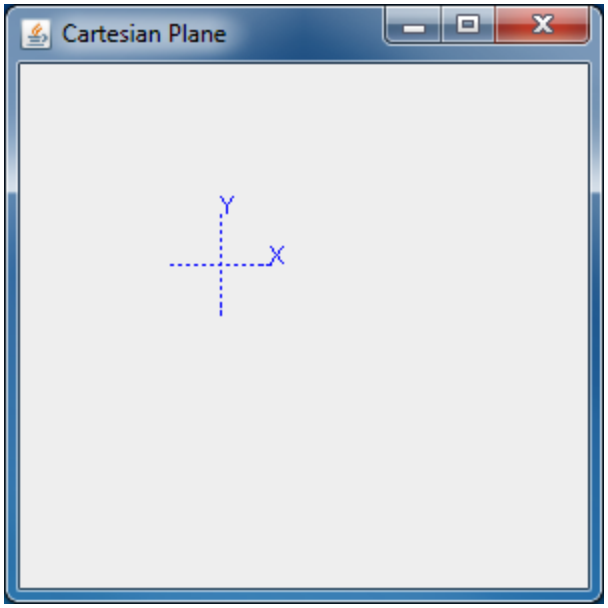
Instructor: Prasun Dewan

Prerequisites

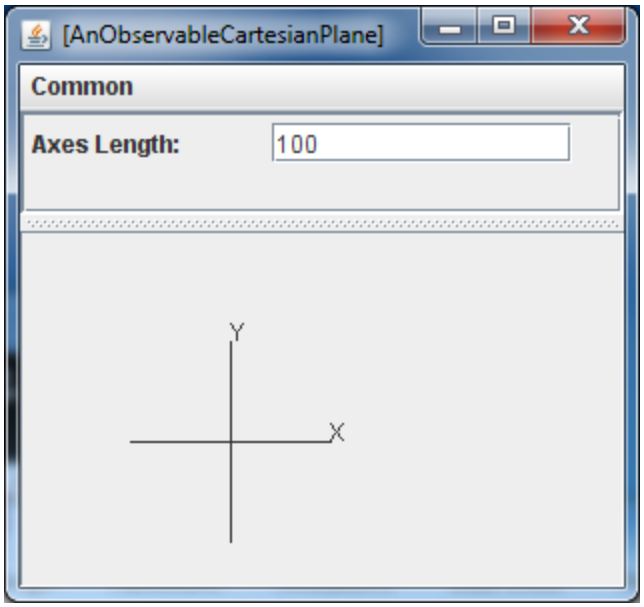
- MVC and Toolkits
- Composite Object Shapes

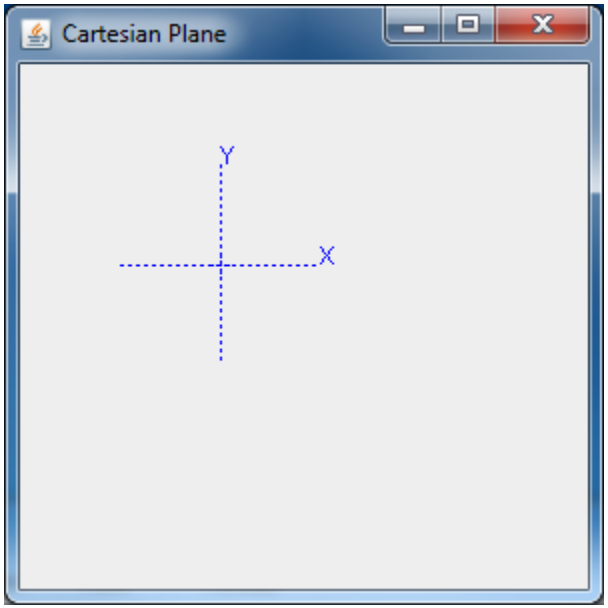
Cartesian Plane: Extra Manual View



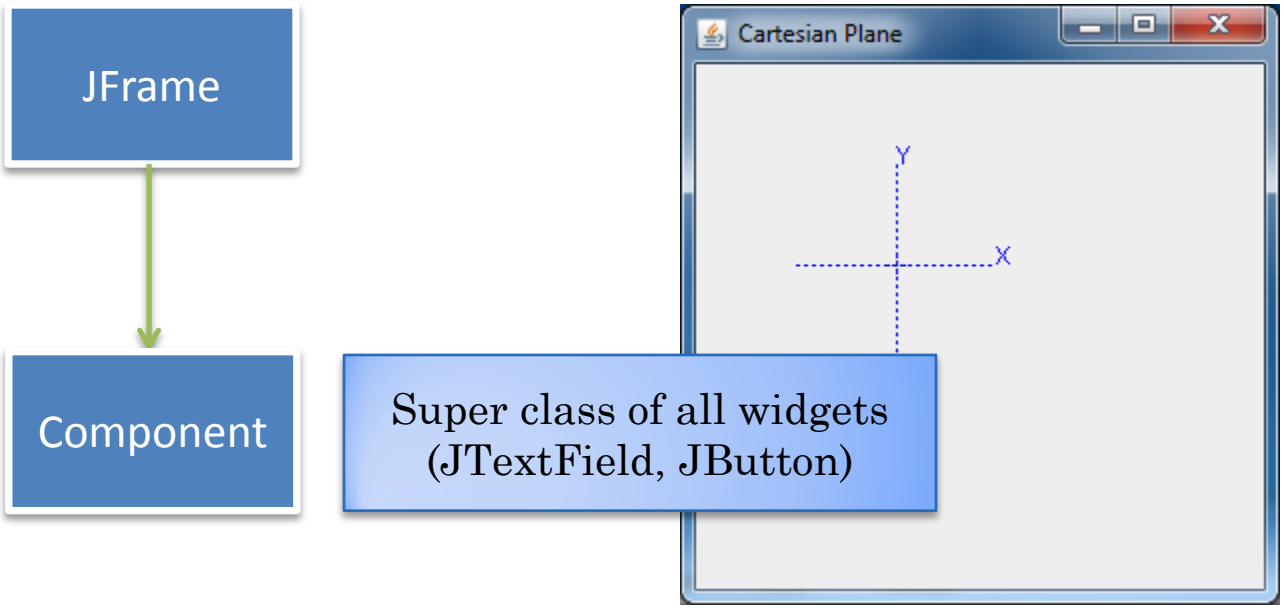


Synchronized Extra Manual View

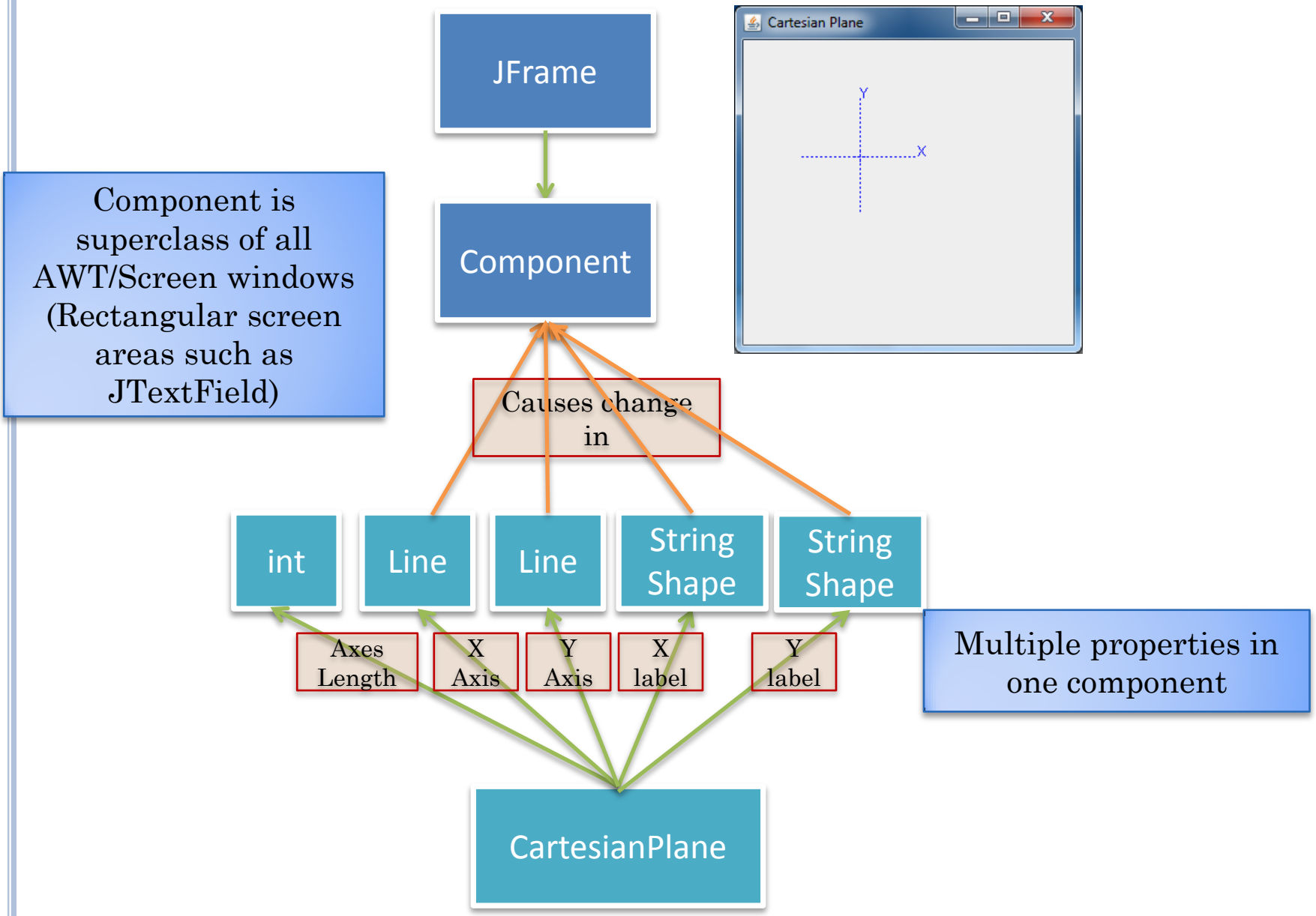




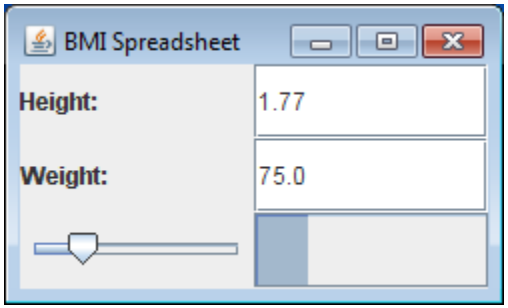
Logical Structure

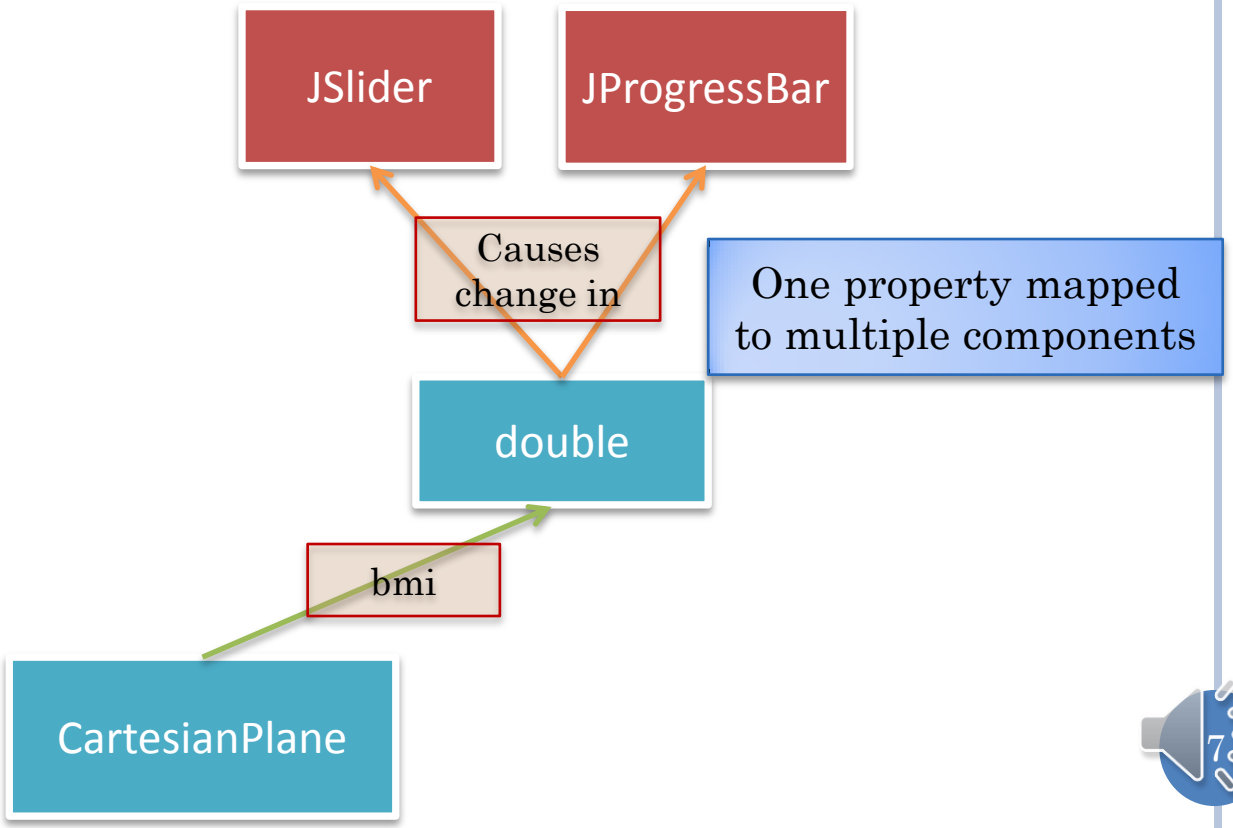


Multiple Properties→One View

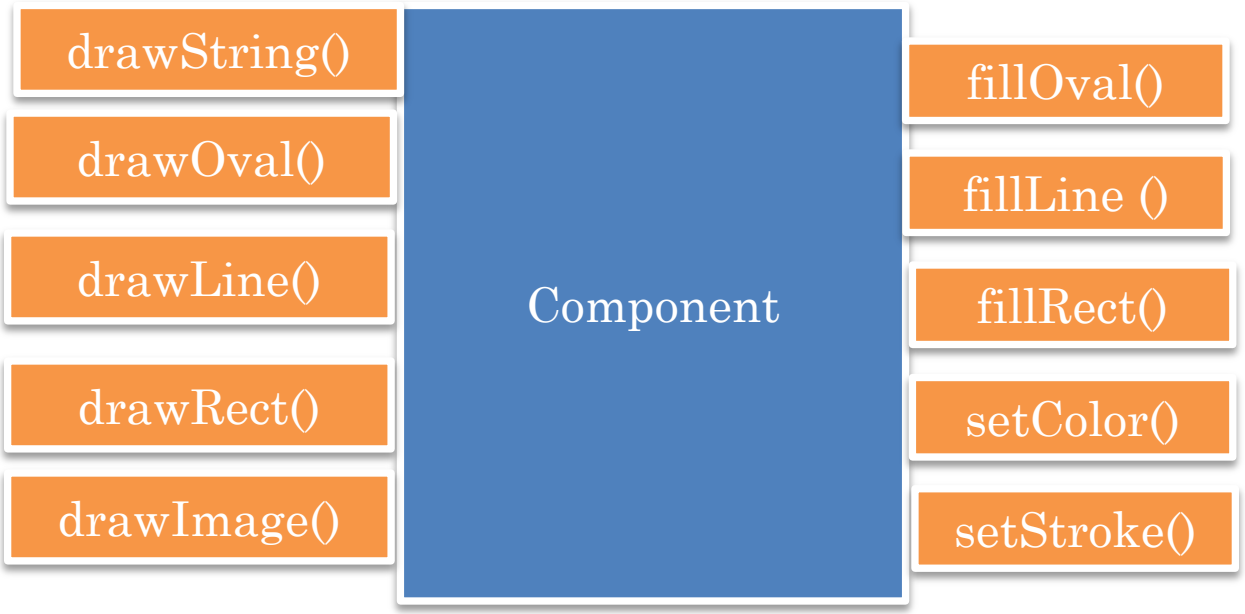


Dual: One Property→Multiple Views



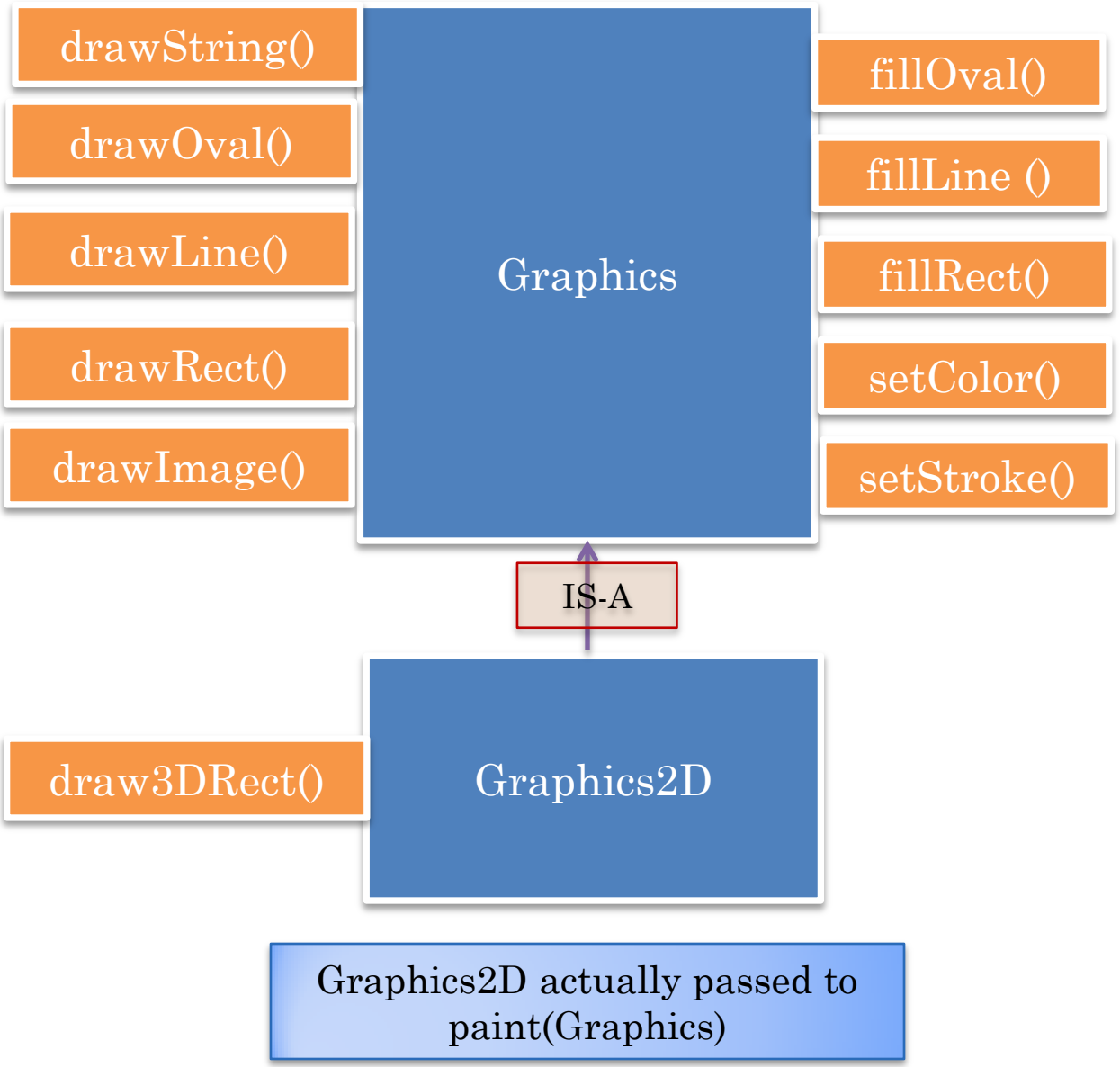


Component API?



Need some object that encapsulates a component and a pen

Graphics Context (Pen + Component)



Use of Graphics

```
public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    g2.setStroke(dotted);
    g.setColor(Color.BLUE);
    draw(g, cartesianPlane);
}
public static void draw(Graphics g,
        CartesianPlane aCartesianPlane)
    draw(g, aCartesianPlane.getXLabel());
    draw(g, aCartesianPlane.getYLabel());
    draw(g, aCartesianPlane.getXAxis());
    draw(g, aCartesianPlane.getYAxis());
  }
  public static void draw(Graphics g, Line aLine) {
    g.drawLine(aLine.getX(), aLine.getY(), aLine.getX() +
                aLine.getWidth(), aLine.getY() + aLine.getHeight());
  }
  public static void draw(Graphics g, Label aLabel) {
    Point location = aLabel.getLocation();
    String s = aLabel.getText();
    g.drawString(s, location.getX(), location.getY());
  }
```

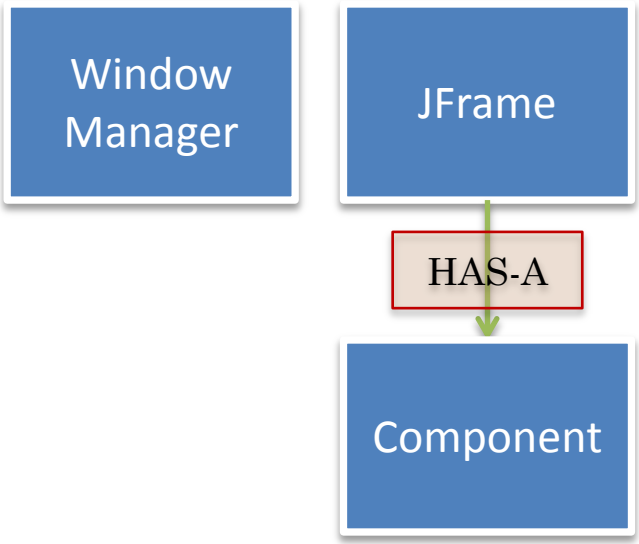
Which component of MVC has this method?

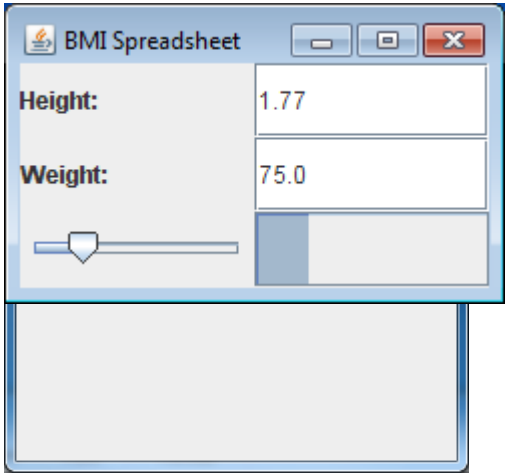
When is it called?

Can we call paint, draw?

How is a Graphics object assigned?

(Re)Painting Responsibility





Who repaints the window: window manager or application

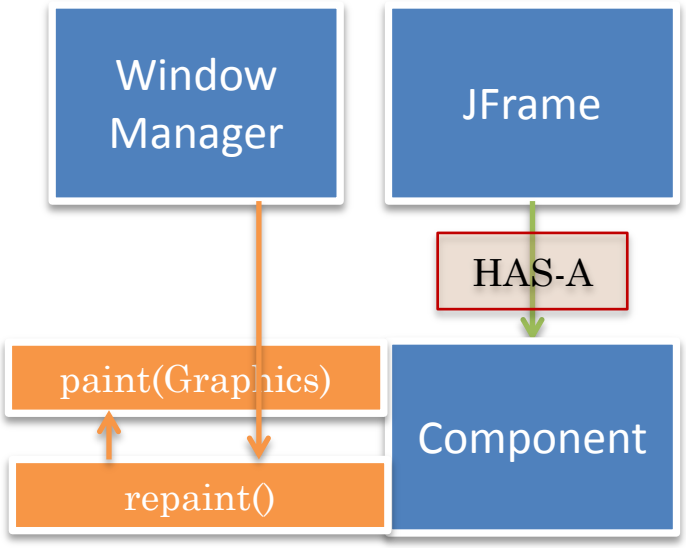
While the component is unexposed, the drawing might have changed.

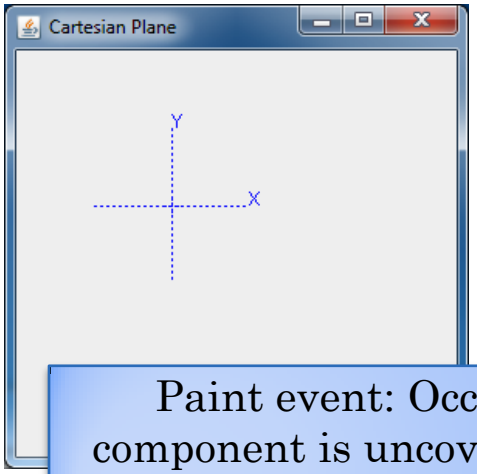
How should application react to window being re-exposed?

So, in general the application must redraw it.

How can we combine code for drawing and redrawing when exposed

Paint Event: What the Window System Does





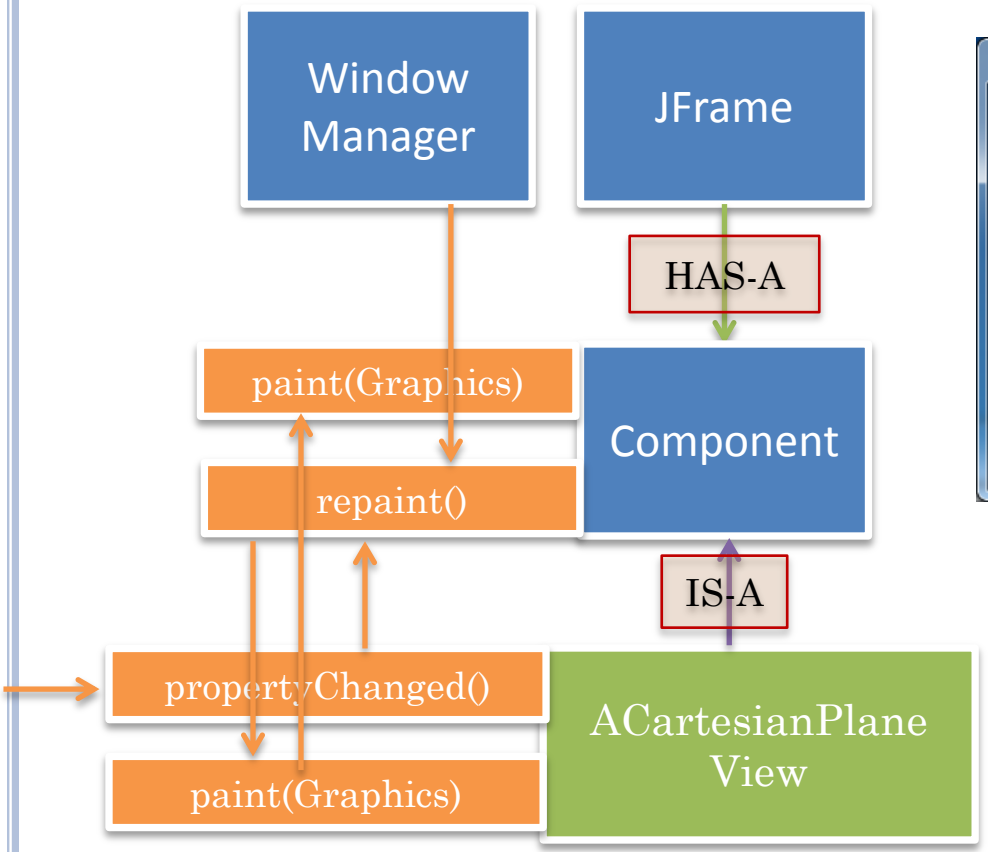
Paint event: Occurs when a component is uncovered (exposed) or when it is first displayed

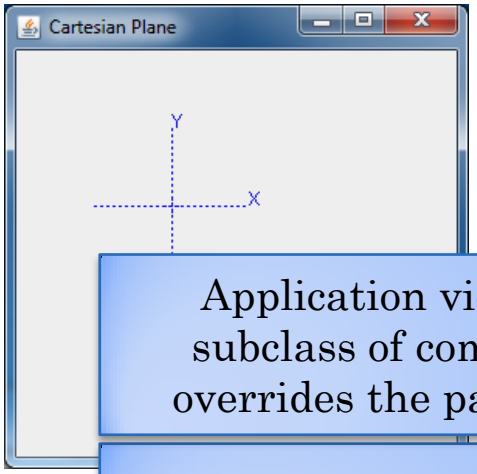
Paint event causes the repaint() method of the associated component to be called so that the component can (re)draw itself

Repaint calls paint passing it the graphics object representing the component and pen.

How to make application view (re) draw graphics in component?

The Inheritance Approach in Swing/AWT





Application view becomes subclass of component and overrides the paint() method

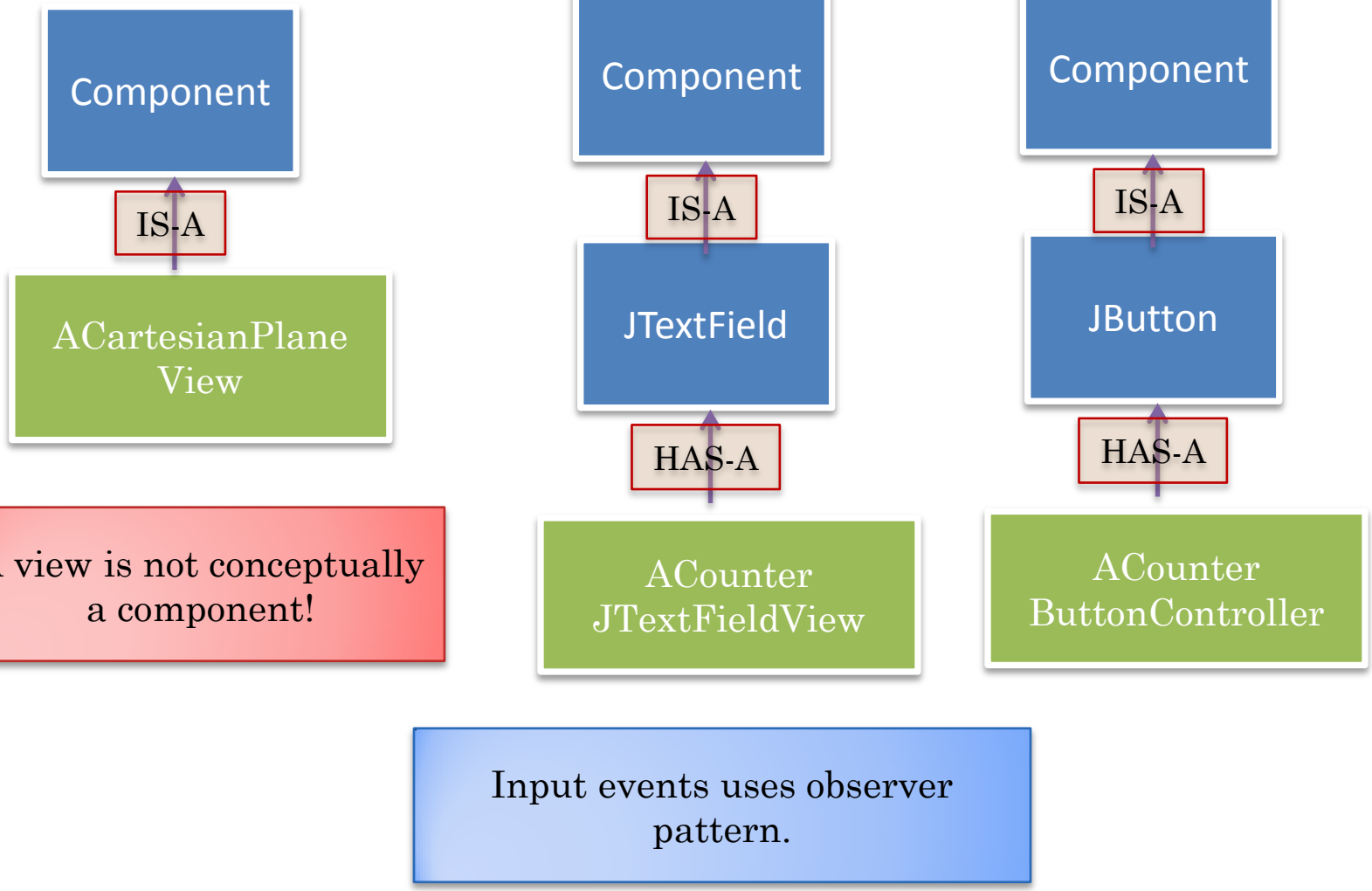
In the paint method it does the view task of displaying the shapes.

On receiving a property change it calls repaint since it cannot manufacture a graphics context

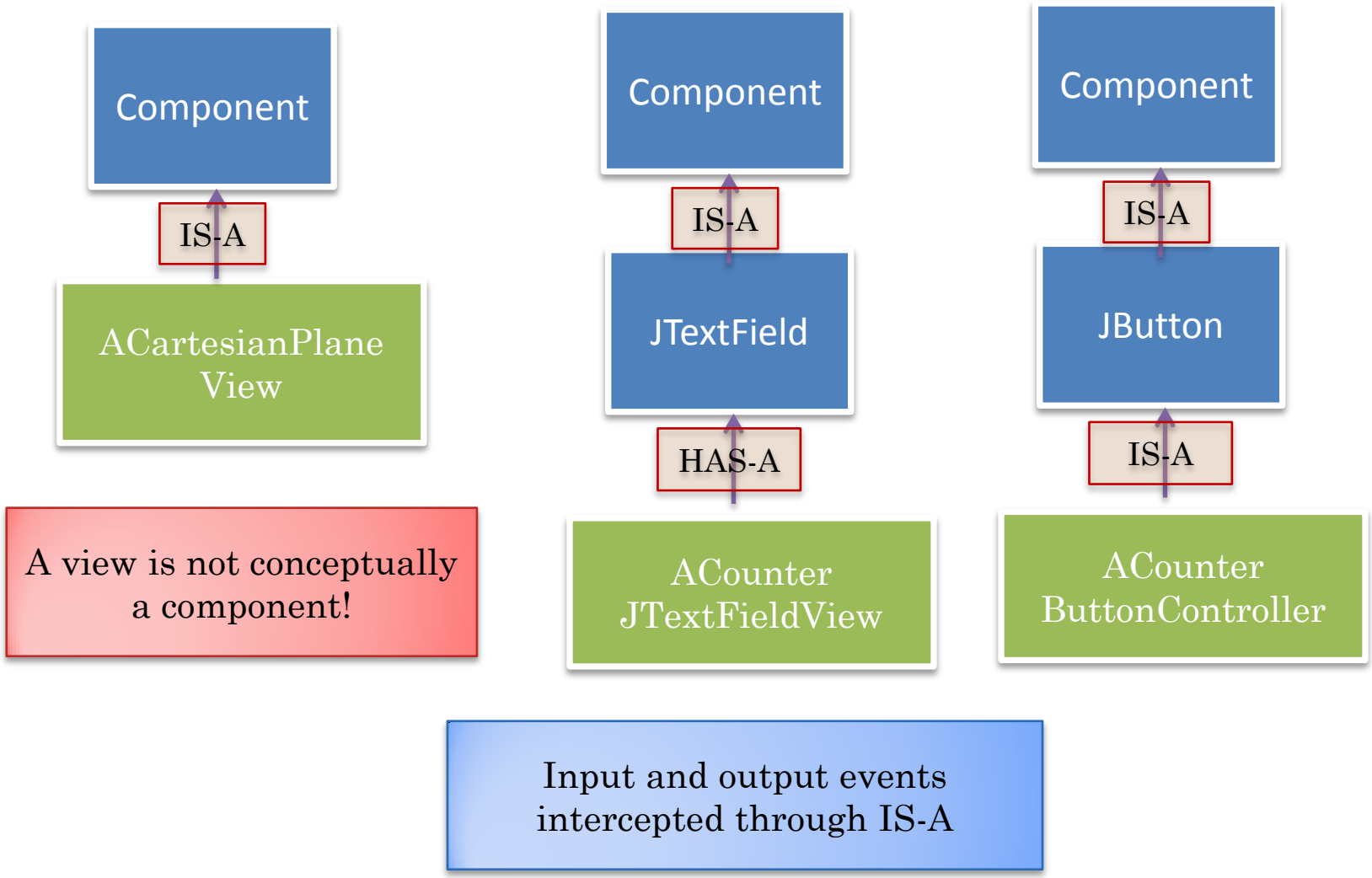
SWT uses observer pattern.

A view is not conceptually a component!

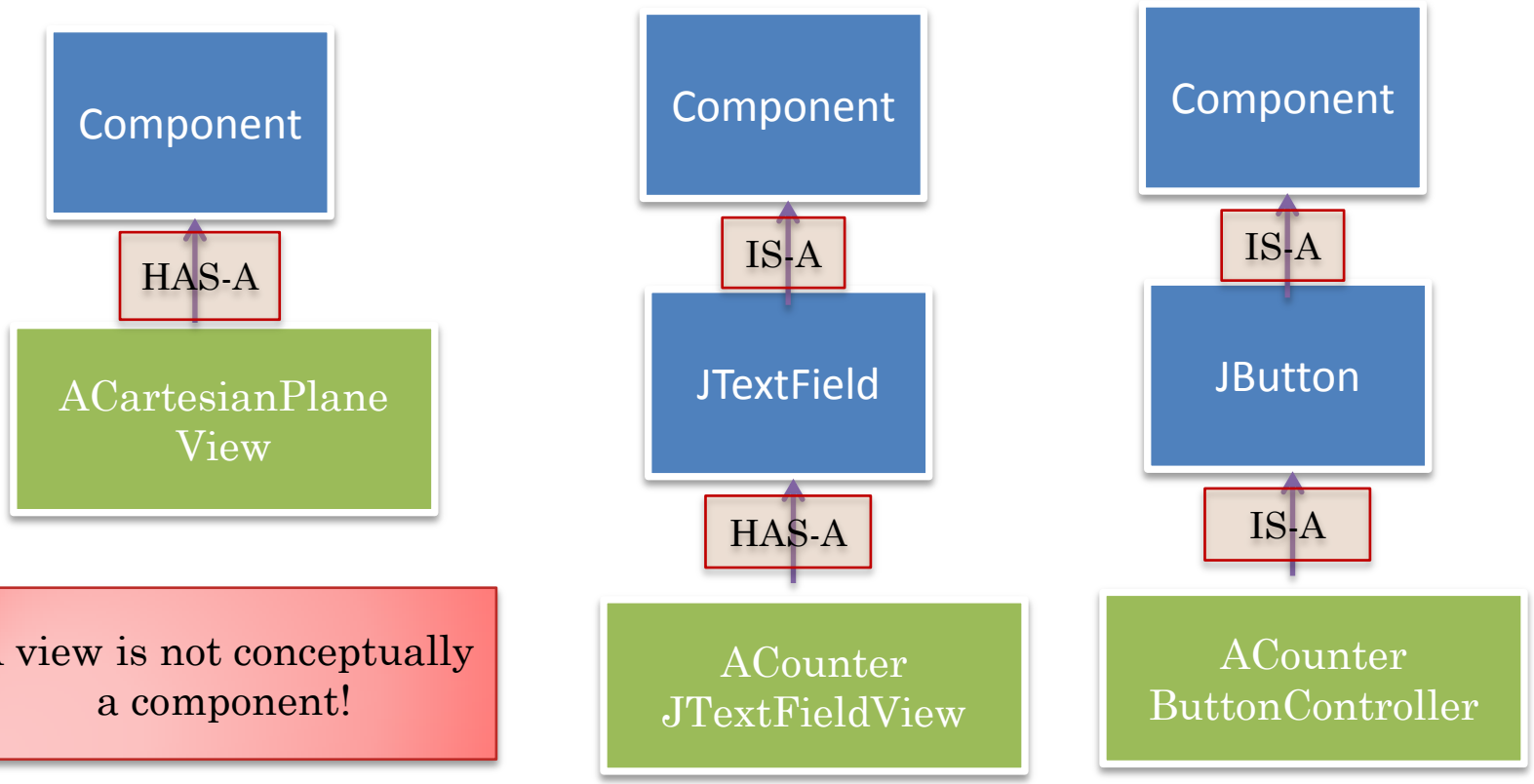
IS-A vs. HAS-A (1.1)



IS-A vs. HAS-A Event Interception (1.0)



Eclipse SWT Event Interception



SWT uses observer pattern.

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));
  }
  …
}
```

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));
}
```

Overriding Paint Method

```
public class ACartesianPlaneView extends Component
                implements PropertyChangeListener {
  BasicStroke dotted = new BasicStroke(1f, sicStroke.CAP_ROUND,
                BasicStroke.JOIN_ROUND, 1f, new float[] {2f}, 0f);
  CartesianPlane cartesianPlane;
  public ACartesianPlaneView(
                CartesianPlane aCartesianPlane) {
    cartesianPlane = aCartesianPlane;
  }
  public void propertyChange(PropertyChangeEvent evt) {
    repaint(); // causes paint to be called
  }
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    g2.setStroke(dotted);
    g.setColor(Color.BLUE);
    draw(g, cartesianPlane);
  }
```

Drawing

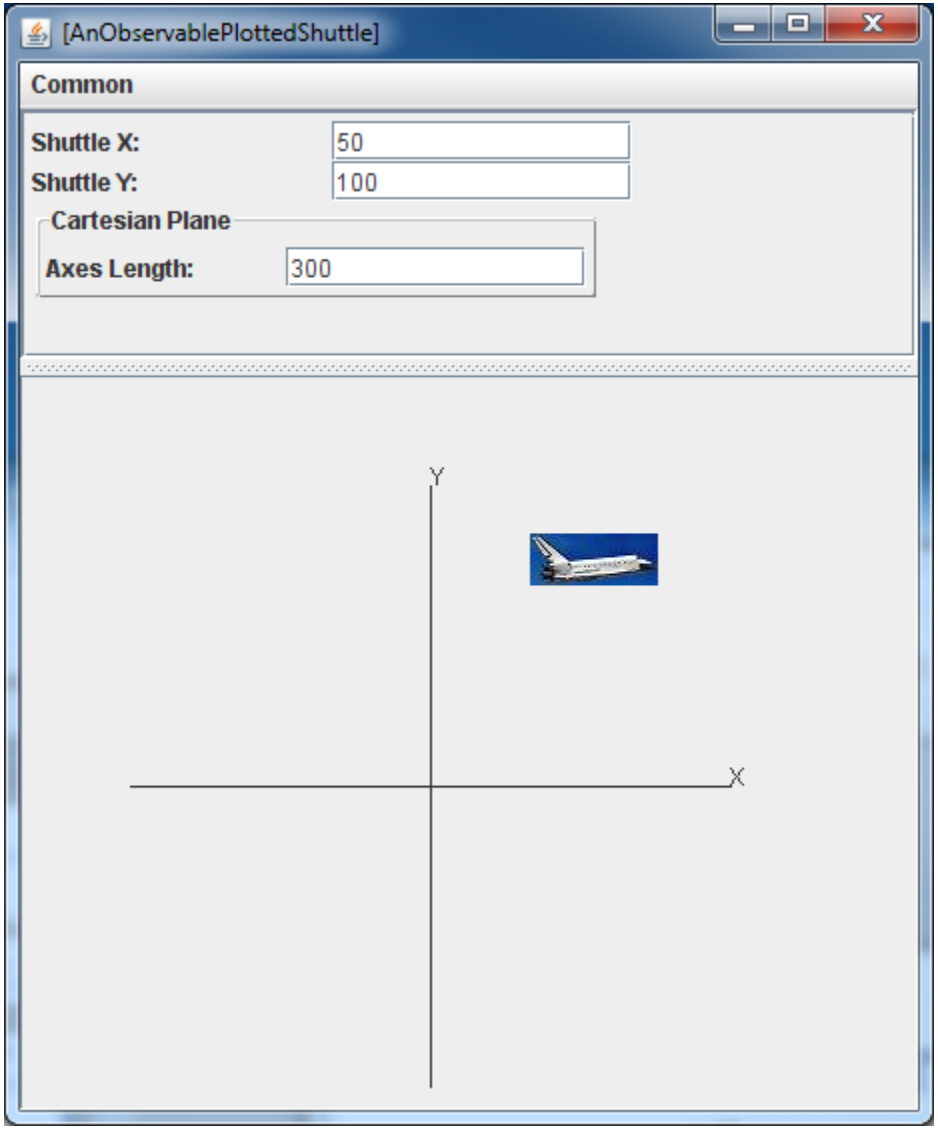
```
public static void draw(Graphics g,
      CartesianPlane aCartesianPlane) {
  draw(g, aCartesianPlane.getXLabel());
  draw(g, aCartesianPlane.getYLabel());
  draw(g, aCartesianPlane.getXAxis());
  draw(g, aCartesianPlane.getYAxis());
}
public static void draw(Graphics g, Line aLine) {
  g.drawLine(aLine.getX(), aLine.getY(), aLine.getX() +
             aLine.getWidth(), aLine.getY() + aLine.getHeight());
}
public static void draw(Graphics g, Label aLabel) {
  Point location = aLabel.getLocation();
  String s = aLabel.getText();
  g.drawString(s, location.getX(), location.getY());
}
```

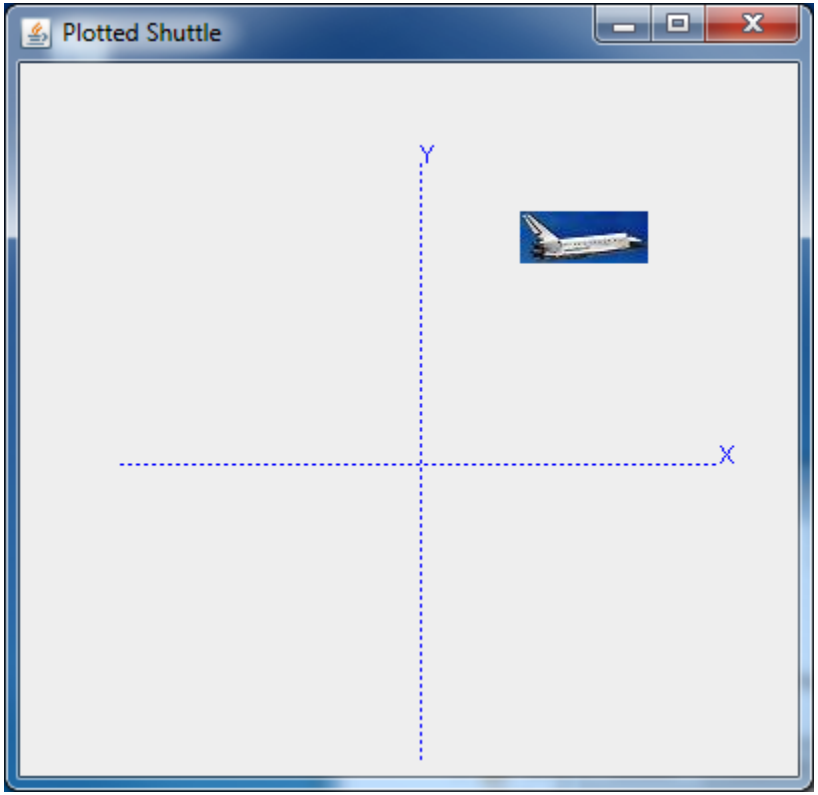
Cartesian Plane Composer

```
public class CartesianPlaneComposer {
  public static void main(String[] args) {
  CartesianPlane cartesianPlane =
      new AnObservableCartesianPlane(100, 100, 100);
  PropertyChangeListener view = new
      ACartesianPlaneView(CartesianPlane);
  cartesianPlane.addPropertyChangeListener(view);
  ((ObservableLine) cartesianPlane.getXAxis()).
     addPropertyChangeListener(view);
  ((ObservableLine)
     cartesianPlane.getYAxis()).addPropertyChangeListener(view);
  ((ObservableStringShape)
    cartesianPlane.getXLabel()).addPropertyChangeListener(view);
  ((ObservableStringShape)
    cartesianPlane.getYLabel()).addPropertyChangeListener(view);
  JFrame frame = new JFrame(" Cartesian Plane");
  frame.add((Component) view);
  frame.setSize(300, 300);
  frame.setVisible(true);
  ObjectEditor.edit(CartesianPlane);
  }
}
```

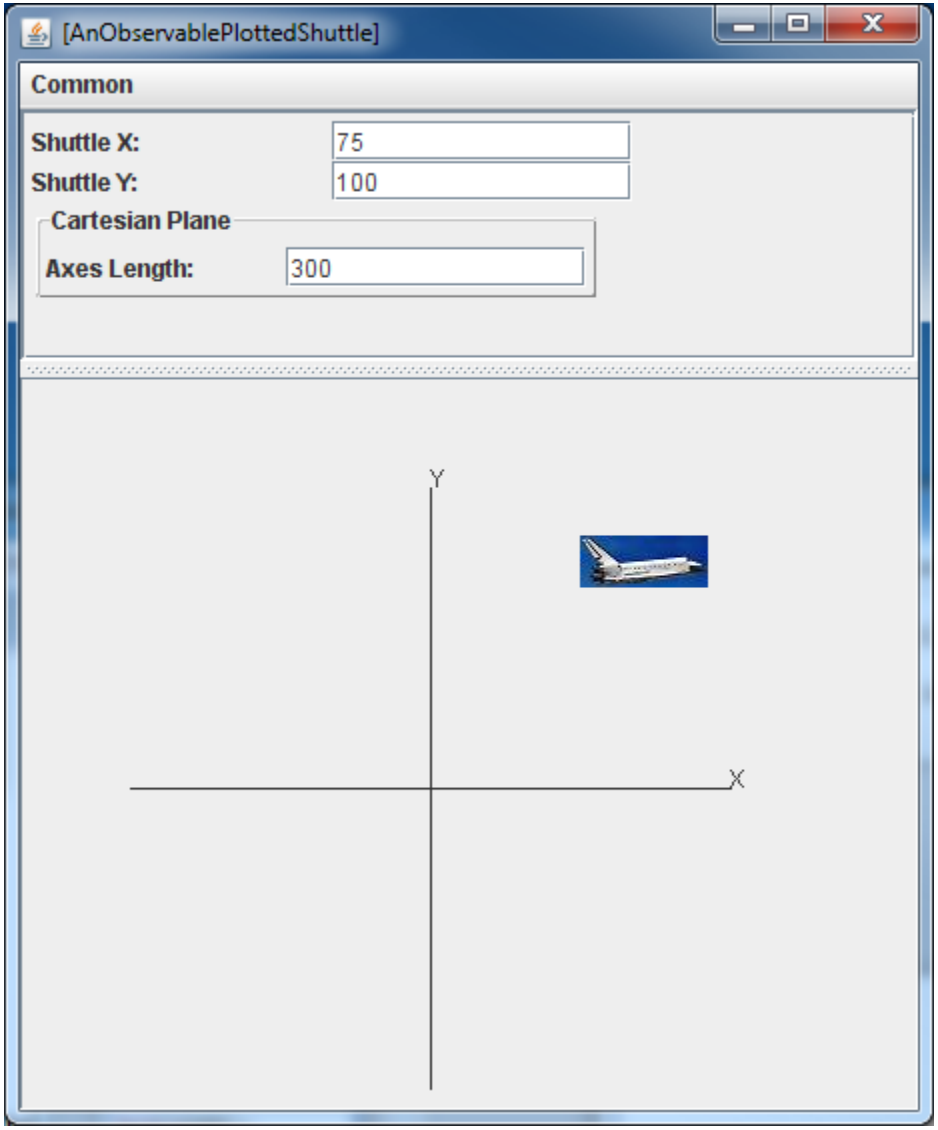
Making view a listener not only of model but also all of its components though not needed in this problem

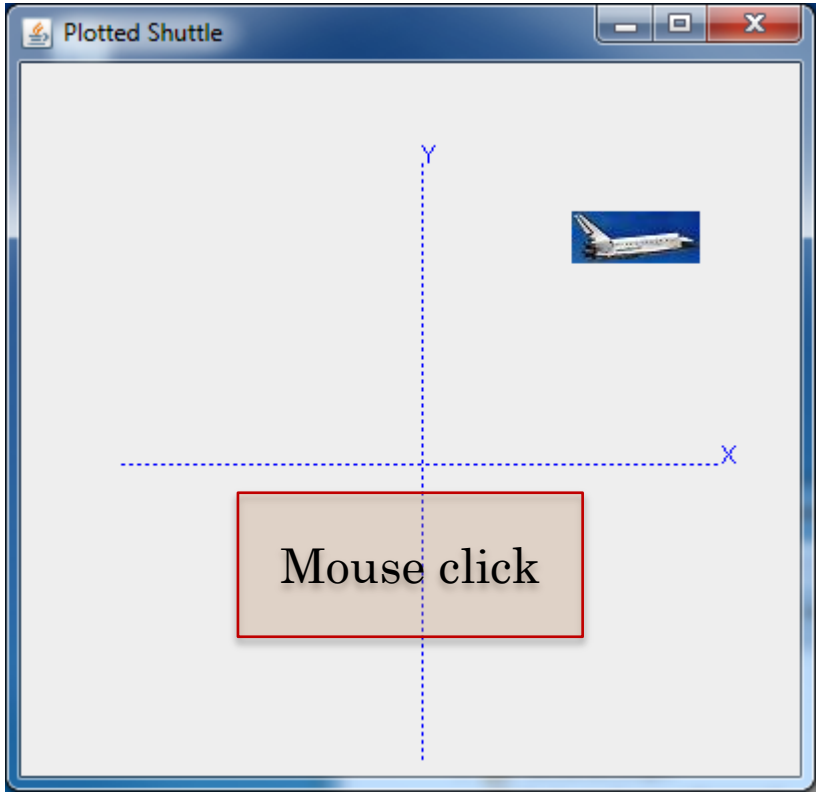
Plotted Shuttle



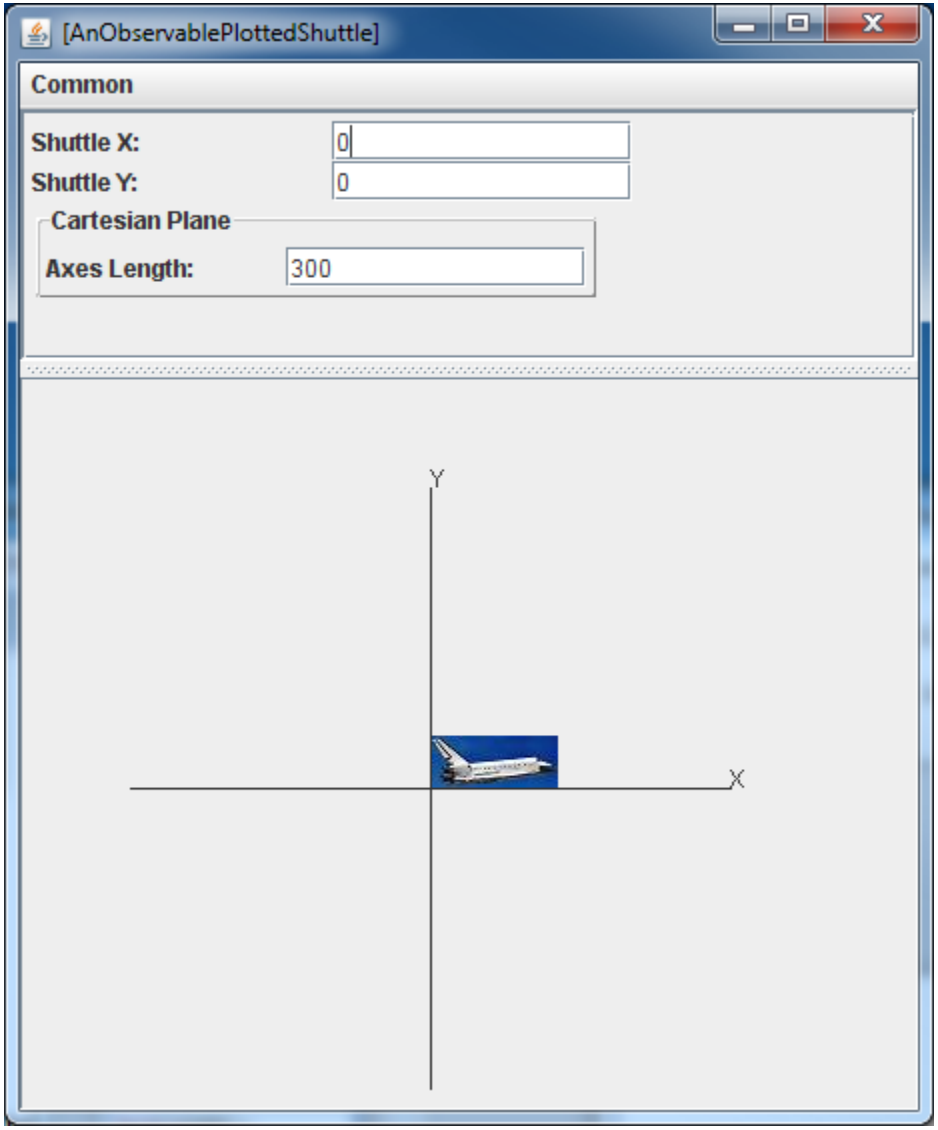


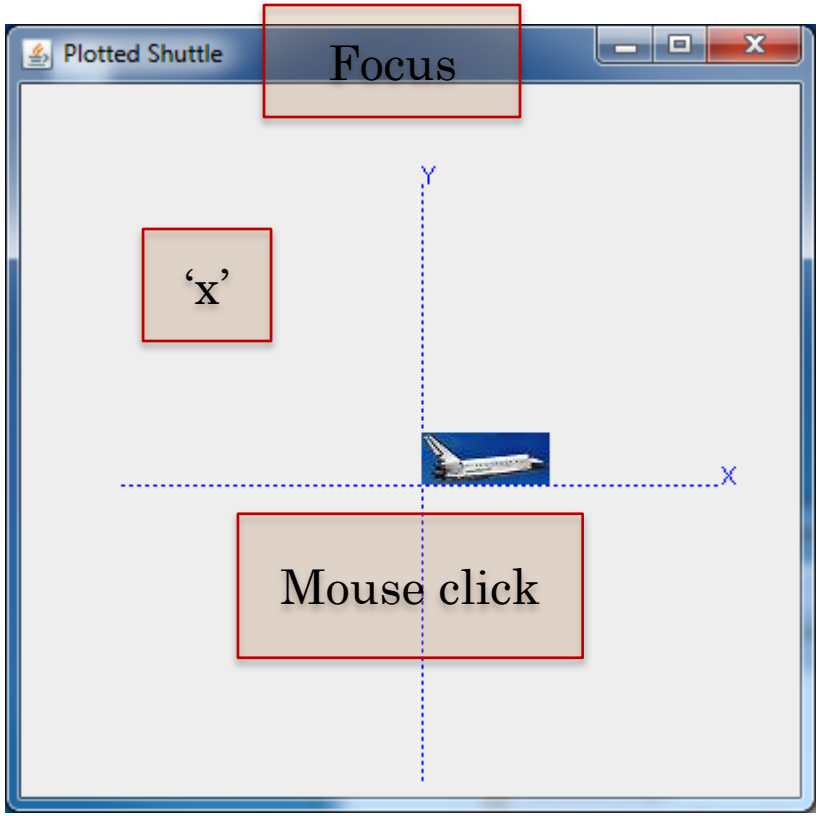
User Edits Shuttle X in OE Window





User Clicks Mouse in Manual Display

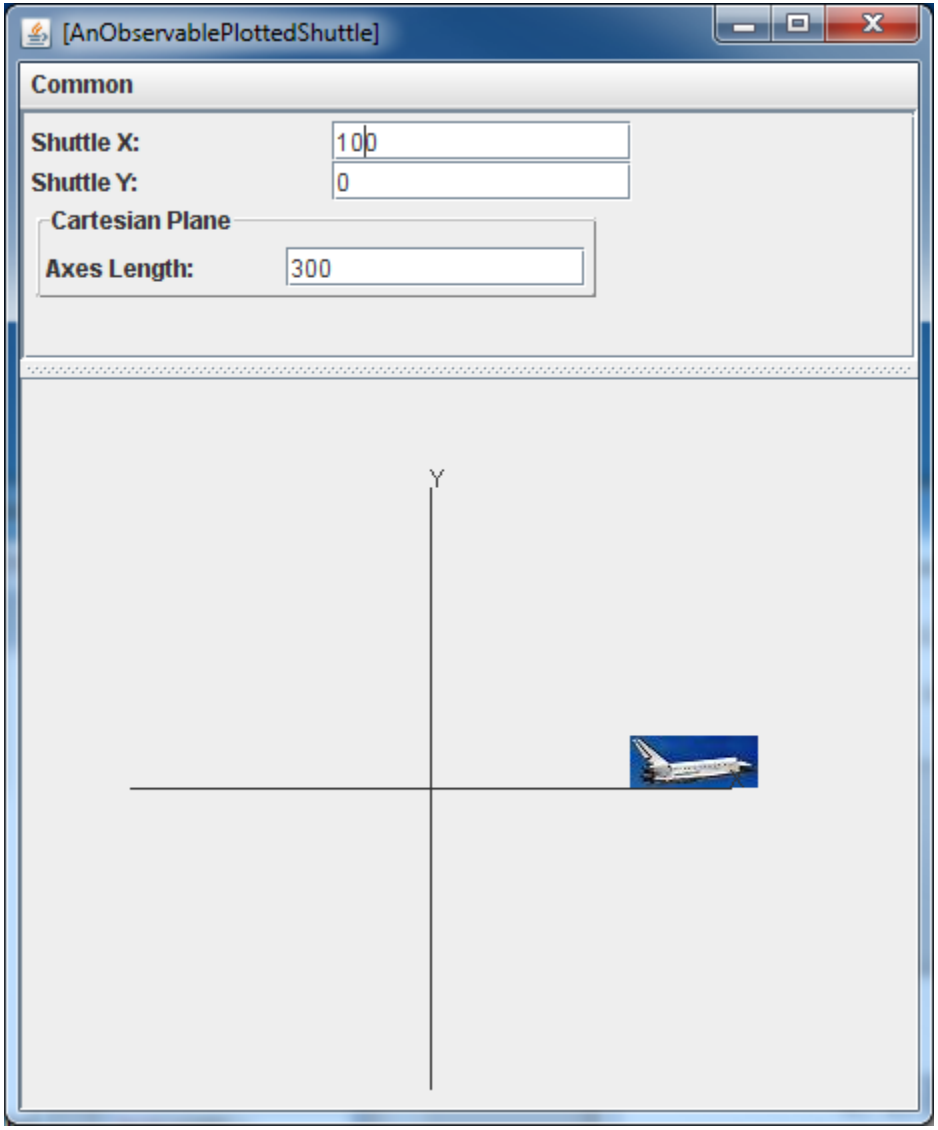


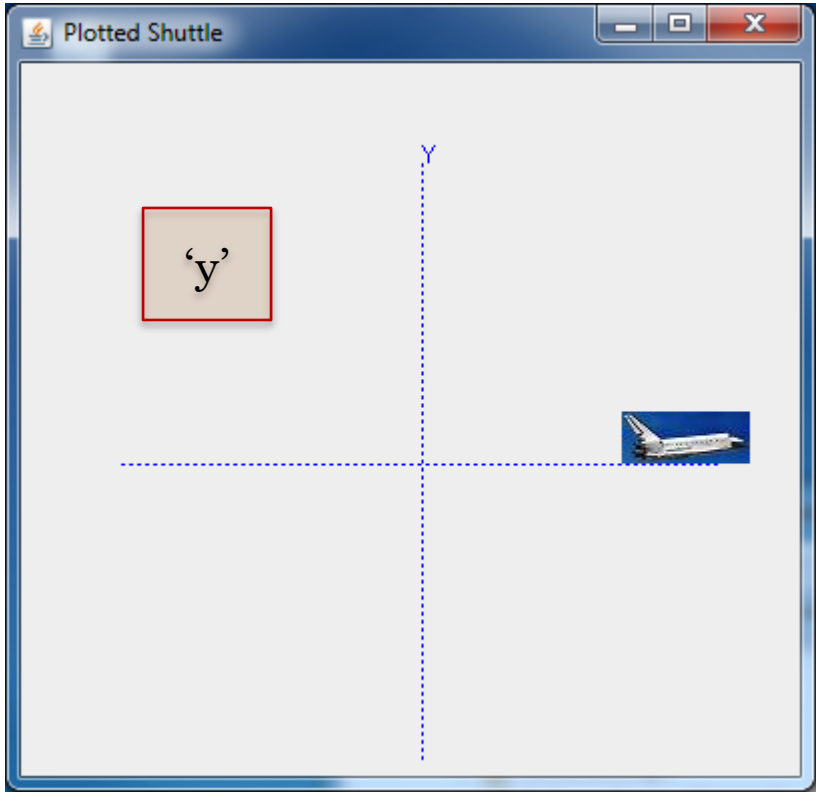


Focus on window (Frame) by clicking in its border in Windows

Focus window is the one that gets keyboard events

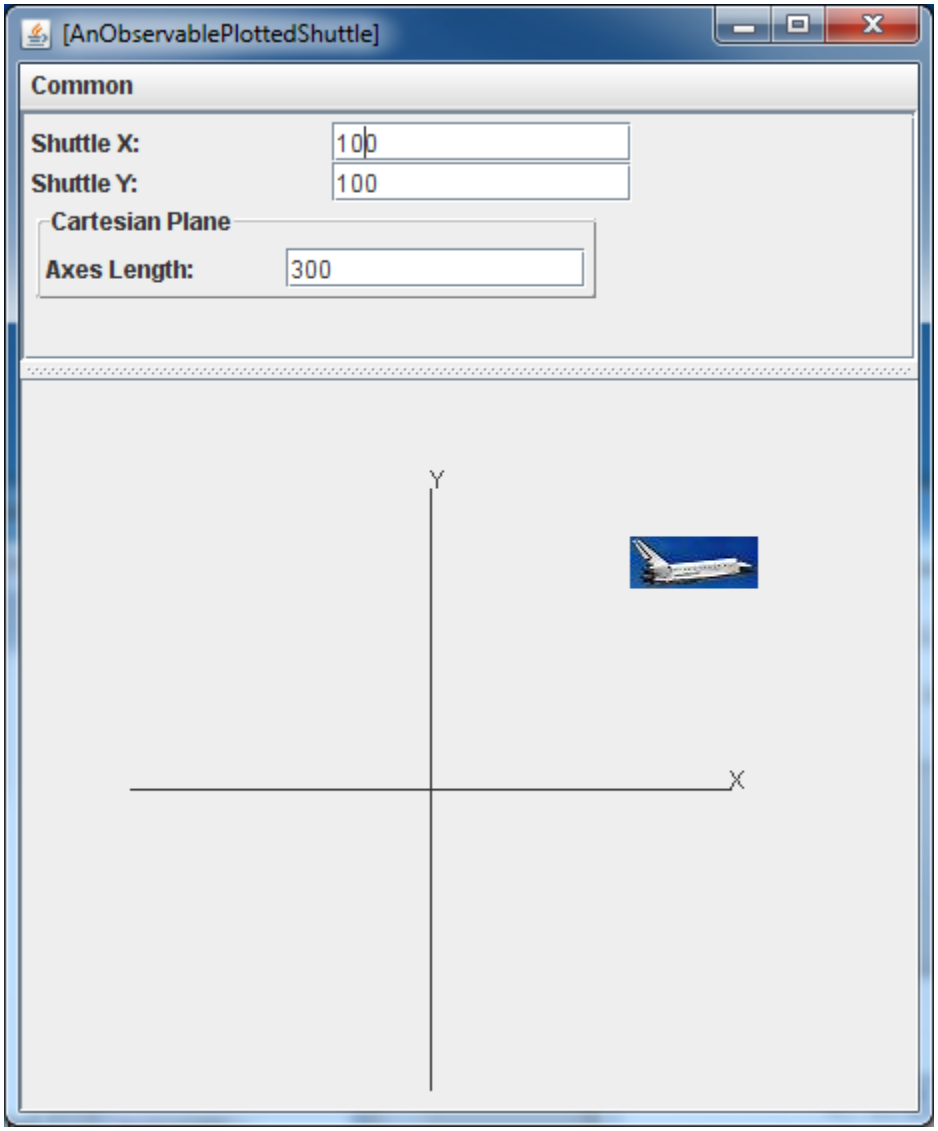
Result of Typing 'x' in Manual Display

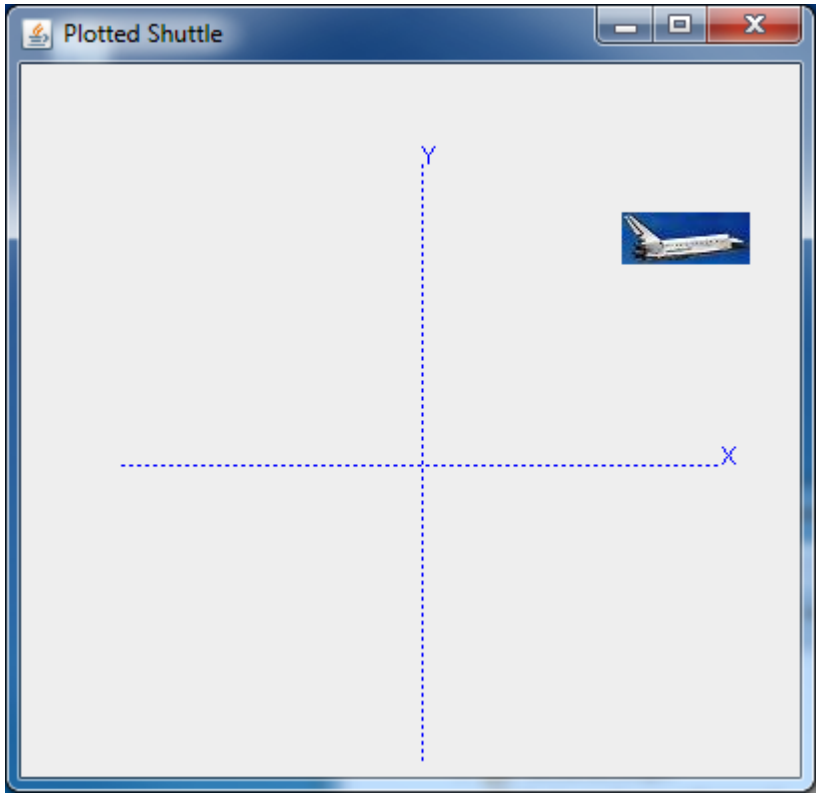




Shuttle goes to some preset X coordinate

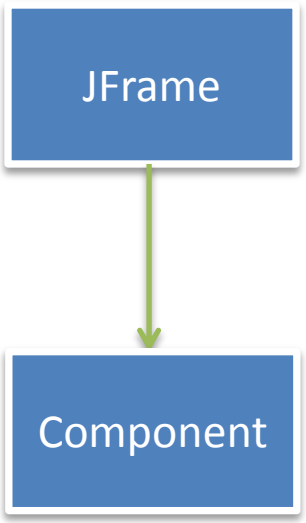
Result of 'Y' in Manual Display

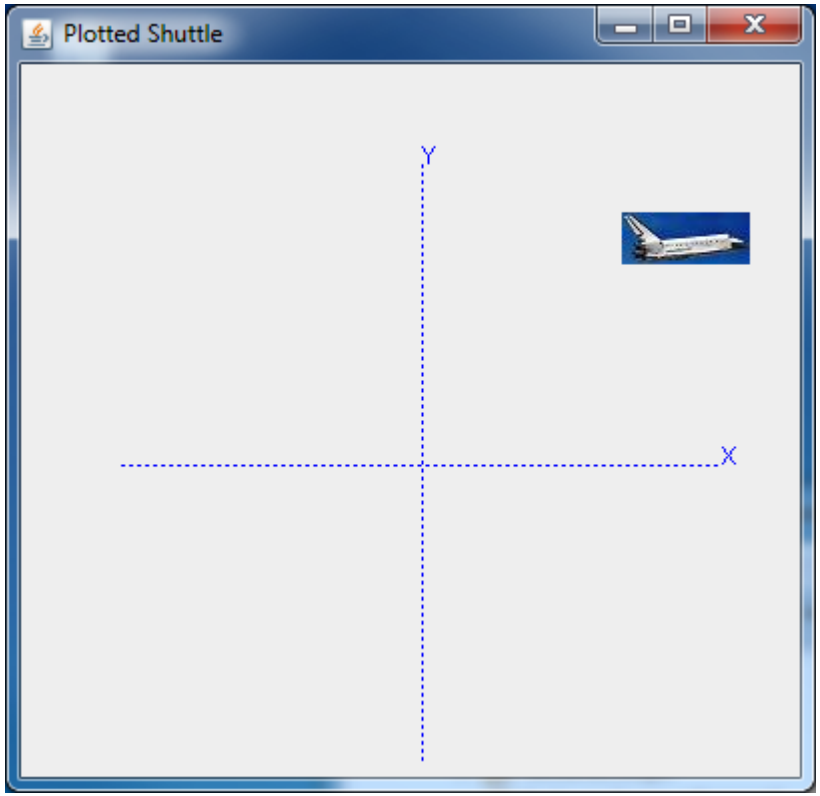




Shuttle goes to some preset Y coordinate

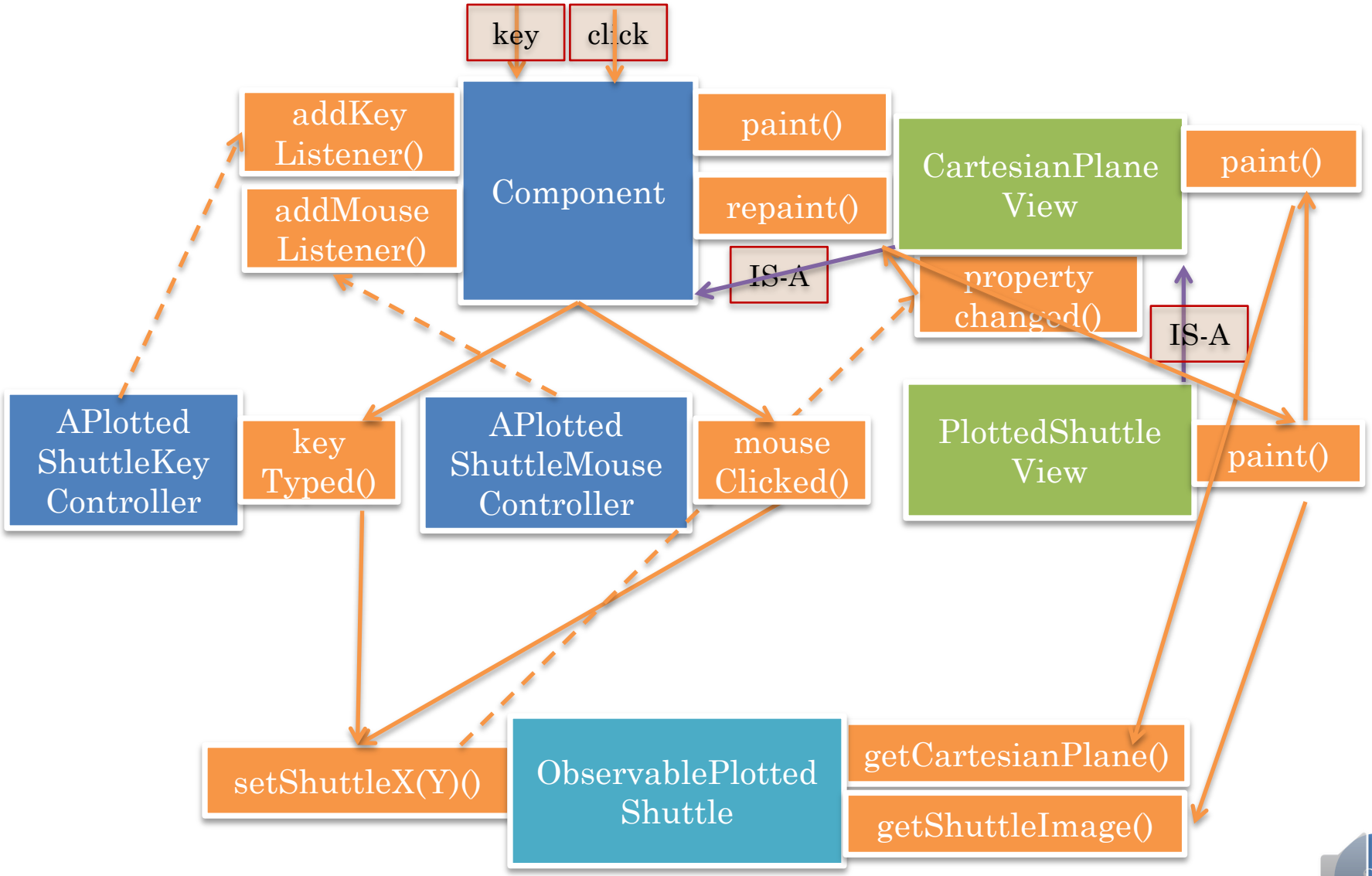
Logical Structure





Re-use of view?

Plotted Shuttle MVC



Reusing View and Drawing Images

```
public class APlottedShuttleView
             extends ACartesianPlaneView implements BeanView {
ObservablePlottedShuttle plottedShuttle;
  public APlottedShuttleView (
             ObservablePlottedShuttle aPlottedShuttle) {
   super((ObservableCartesianPlane) aPlottedShuttle.getCartesianPlane());
   plottedShuttle = aPlottedShuttle;
}
}
```

Reusing View and Drawing Images

```
  public void paint(Graphics g) {
    super.paint(g);
    Graphics2D g2 = (Graphics2D) g;
    draw(g2, plottedShuttle.getShuttleImage());
  }
  public  void draw(Graphics2D g, ShuttleImage anImage) {
    Image img =
          Toolkit.getDefaultToolkit().getImage(anImage.getImageFileName());
    g.drawImage(img, anImage.getX(), anImage.getY(), this);
   }
  public  void draw(Graphics2D g, PlottedShuttle aShuttleLocation) {
    draw(g, aShuttleLocation.getCartesianPlane());
    draw (g, aShuttleLocation.getShuttleImage());
   }
}
```

Shuttle image drawn over the Cartesian plane

Shuttle or Cartesian plane on top?

Mouse Controller

```
public class APlottedShuttleMouseController implements MouseListener {
  Component shuttleComponent;
  PlottedShuttle plottedShuttle;
  public APlottedShuttleMouseController (PlottedShuttle aPlottedShuttle,
                                         Component aShuttleComponent) {
    shuttleComponent = aShuttleComponent;
    plottedShuttle = aPlottedShuttle;
    shuttleComponent.addMouseListener(this);
  }
  public void mouseClicked(MouseEvent e) {
    plottedShuttle.setShuttleX(0);
    plottedShuttle.setShuttleY(0);
  }
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {}
  public void mouseReleased(MouseEvent e) {}
}
```

Key Controller

```
public class APlottedShuttleKeyController implements KeyListener {
  Component shuttleComponent;
  PlottedShuttle plottedShuttle;
  public static final int TARGET_X = 100;
  public static final int TARGET_Y = 100;
  public APlottedShuttleKeyController (PlottedShuttle aPlottedShuttle,
                                       Component aShuttleComponent) {
    shuttleComponent = aShuttleComponent;
    plottedShuttle = aPlottedShuttle;
    shuttleComponent.addKeyListener(this);
    shuttleComponent.setFocusable(true);
  }
}
```

Key Controller

```
public void keyTyped(KeyEvent e) {
   char typedChar = e.getKeyChar();
   switch (typedChar) {
     case 'x' :
       plottedShuttle.setShuttleX(TARGET_X);
       break;
     case 'y':
       plottedShuttle.setShuttleY(TARGET_Y);
       break;
   }
 }
 public void keyPressed(KeyEvent e) {}
 public void keyReleased(KeyEvent e) {}
```

Plotted Shuttle Composer

```
public class PlottedShuttleComposer {
  public static void main(String[] args) {
    ObservablePlottedShuttle plottedShuttle = new
           AnObservablePlottedShuttle(50, 100);
    PropertyChangeListener view = new APlottedShuttleView(plottedShuttle);
    plottedShuttle.addPropertyChangeListener(view);
    MouseListener mouseController =
      new APlottedShuttleMouseController(plottedShuttle, (Component) view);
    KeyListener keyController =
      new APlottedShuttleKeyController(plottedShuttle, (Component) view);
    JFrame frame = new JFrame("Plotted Shuttle");
    frame.add((Component) view);
    frame.setSize(300, 300);
    frame.setVisible(true);
    ObjectEditor.edit(plottedShuttle);
    plottedShuttle.setShuttleY(100);
    plottedShuttle.setShuttleX(50);
  }
}
```

In general should make view listener of all model components to which it listens

Window System, Toolkit and Window Manager

- Window system
  - Supports hierarchical rectangular areas on screens called windows with no predefined behavior
- Toolkit
  - Extends windows to widgets by adding behavior to them
  - A widget IS-A window
  - Sometimes use widget and window synonymously as Java blurs the distinction
- Window manager
  - Allows manipulation of top-level windows (JFrame or Frame instances in Java)
  - Provides user commands and an API to move, resize, iconify, restore, and focus on (for key board input) windows.
  - Puts decorations around windows for invoking these commands

MVC with Beans and Window Output

- Like MVC with Beans we saw earlier
- One or more properties now displayed in an instance of a window class (Component, Panel, JPanel, Canvas) rather than a widget class (JTextField, TextField)
- A view class is now a subclass of a window class. Earlier a view class had a reference to an instance of a widget class.
- A view overrides the paint() method of the window class to draw the properties using its Graphics argument
- Graphics class provides methods to draw lines, rectanges, ovals ….
- When a view receives a notification from the model class it calls the repaint() method which calls the paint() method

MVC with Window Input and Output

- Like MVC with window output: A view class IS-A Window Class
- A controller now listens to mouse and/or key events of a view (window) which it gets in its constructor
- Calls write methods in the model in response to mouse/key events instead of action events