

# Konzepte objektorientierter Programmierung

**Prof. Dr. Peter Müller**

**Werner Dietl**

Software Component Technology

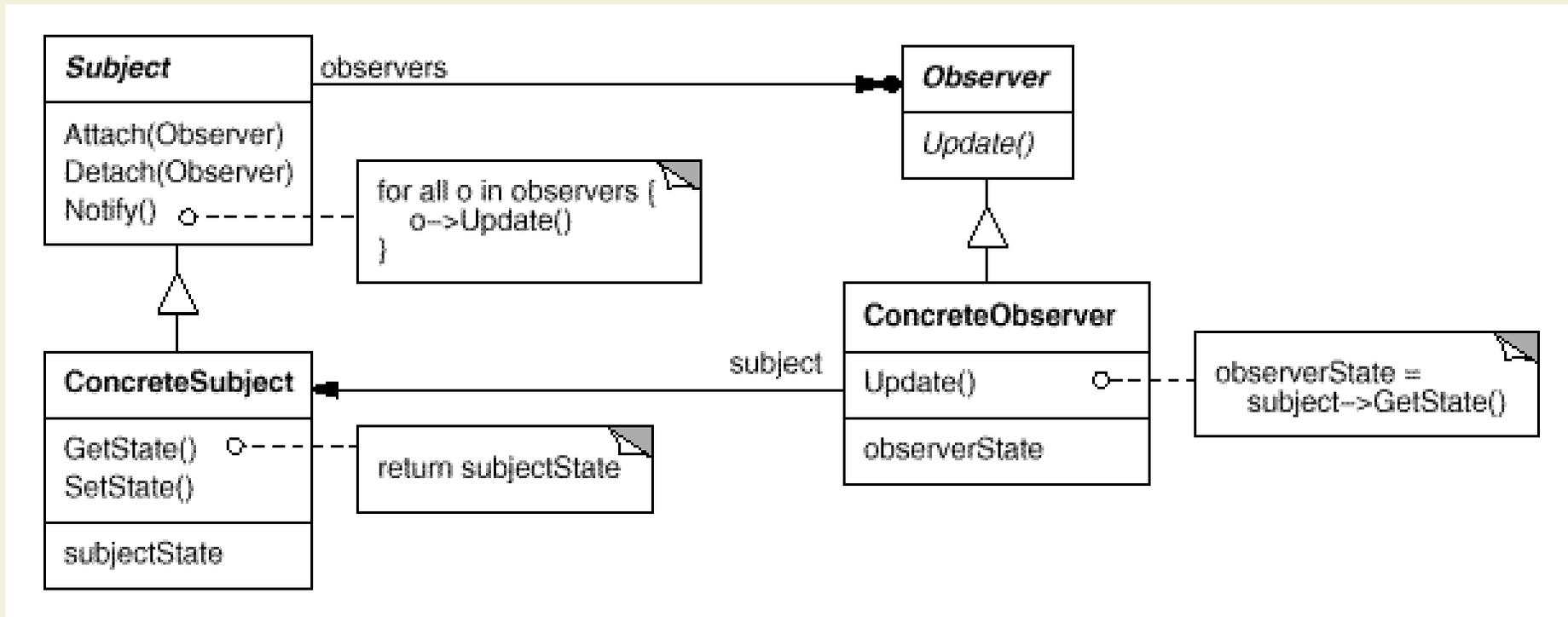
Exercises 5: Frameworks

Wintersemester 06/07

**ETH**

Eidgenössische Technische Hochschule Zürich  
 Swiss Federal Institute of Technology Zürich

# Homework 1 – Observer Pattern



From: Gamma, Helm, Johnson, Vlissides: "Design Patterns"

# Java support for the Observer Pattern

- **Subject** → `java.util.Observable`
  - `void addObserver(Observer o)`
  - `void setChanged()`
  - `void notifyObservers(Object arg)`
- **Observer** → `java.util.Observer`
  - `void update(Observable o, Object arg)`
- **Events** → `java.util.EventObject`
  - `EventObject(Object source)`
  - `Object getSource()`

# Events

```
abstract public class ChangeEvent
  extends java.util.EventObject {

  public ChangeEvent( Object source,
                    int v, int c ) {
    super(source); value = v; change = c; }

  public int getValue() { return value; }

  public int getChange() { return change; }

  private int value;
  private int change;
}
```

# Specialized Events

```
public class IncreaseEvent
  extends ChangeEvent {
  public IncreaseEvent(Object s,int v,int c) {
    super(s, v, c);
  }
}
```

```
public class DecreaseEvent
  extends ChangeEvent {
  public DecreaseEvent(Object s,int v,int c) {
    super(s, v, c);
  }
}
```

## ActiveValue – Setup

```
public class ActiveValue
    extends java.util.Observable {

    private int myval;

    public void run() {
        int change;
        Random crnd = new Random();

        myval = 0;

        while( true ) { ... }
    }
}
```

## ActiveValue – Main Loop

```
while( true ) {  
    change = crnd.nextInt( 11 ) - 5;  
  
    myval += change;  
  
    sendEvent( change );  
  
    Thread.sleep( 1000 );  
}
```

## ActiveValue – Event Notification

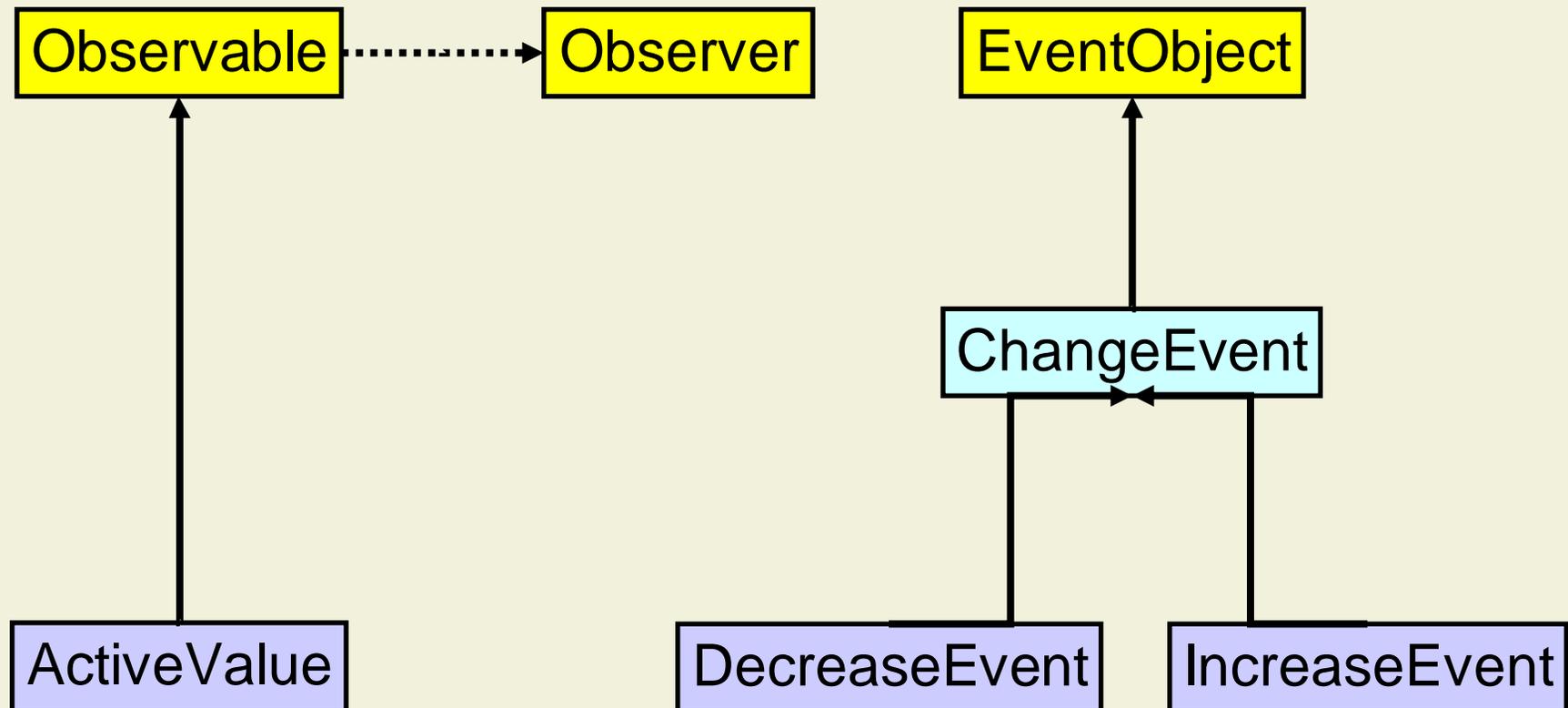
```
public void sendEvent( int change ) {  
    java.util.EventObject e;  
  
    if( change >= 0 ) {  
        e = new IncreaseEvent( this , myval , change ) ;  
    } else {  
        e = new DecreaseEvent( this , myval , change ) ;  
    }  
  
    setChanged( ) ;  
    notifyObservers( e ) ;  
}
```

## ActiveValue – Changing the Value

```
public void reset() {  
    setValue( 0 );  
}
```

```
public void setValue( int value ) {  
    int change;  
  
    change = value - myval;  
    myval = value;  
  
    sendEvent( change );  
}
```

# Class Hierarchy so far



# SignChangeObserver

```
public class SignChangeObserver
    implements java.util.Observer {

    public void update( java.util.Observable o,
                       Object e ) {

        if( !(e instanceof ChangeEvent)) return;
        ChangeEvent ce = (ChangeEvent) e;

        if( ce instanceof IncreaseEvent ) {
            System.out.println( "+" );
        } else {
            System.out.println( "-" );
        }
    }
}
```

# TextChangeObserver

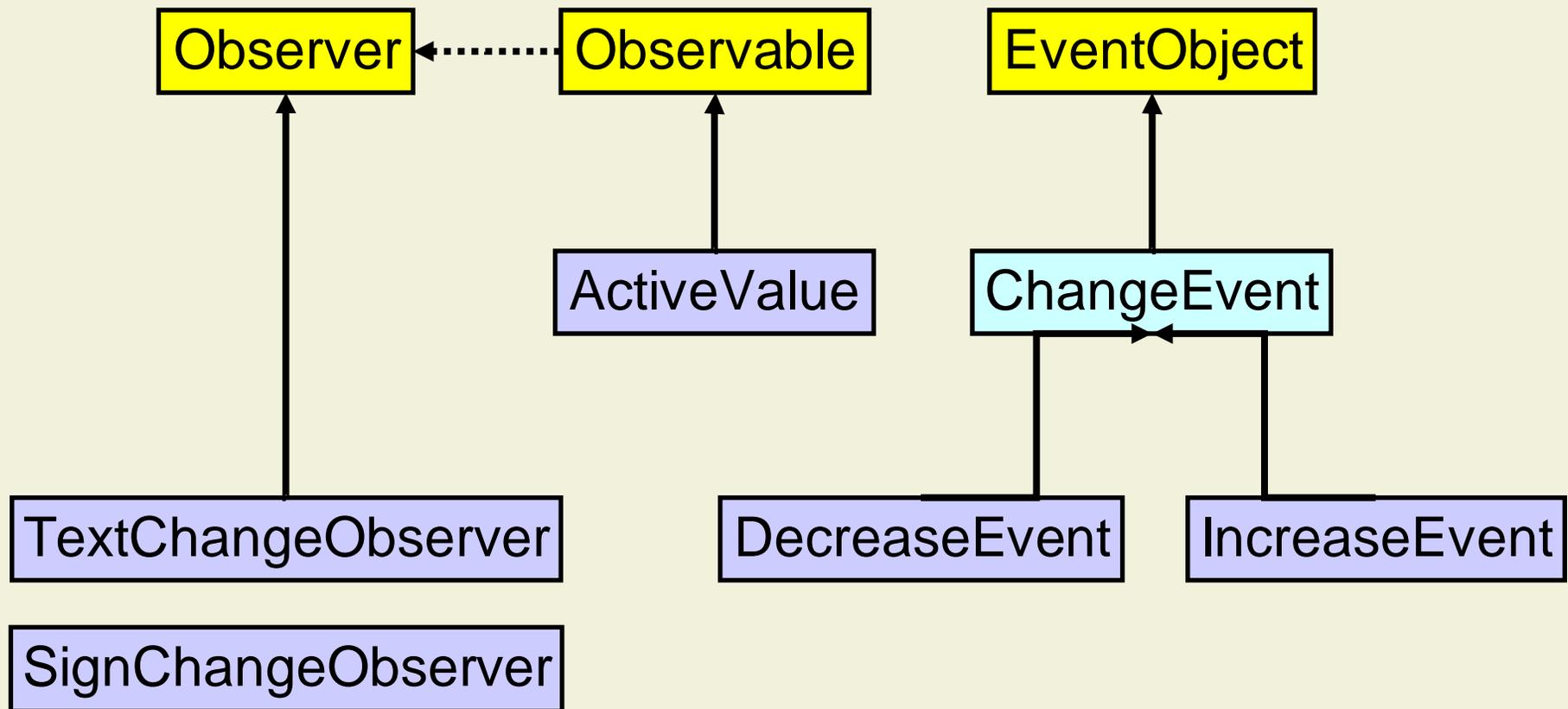
```
public class TextChangeObserver
    implements java.util.Observer {

    public void update(Observable s, Object e) {
        if( !(e instanceof ChangeEvent) ) return;
        ChangeEvent ce = (ChangeEvent) e;
        if( ce instanceof IncreaseEvent ) {
            System.out.print("++++ ");
        } else {
            System.out.print("---- ");
        }
        System.out.println("New Value: "+ce.getValue() +
            " Change: " + ce.getChange() );
    }
}
```

# TextMain

```
public class TextMain {  
    public static void main( String[] args ) {  
        ActiveValue v = new ActiveValue();  
  
        Observer o1 = new TextChangeObserver();  
        Observer o2 = new SignChangeObserver();  
  
        v.addObserver( o1 );  
        v.addObserver( o2 );  
  
        v.run();  
    }  
}
```

# Class Hierarchy



## GuiMain – Adding a Graphical UI

```
public class GuiMain {  
    public static void main( String[] args ) {  
  
        ActiveValue v = new ActiveValue();  
  
        Observer o1 = new GuiChangeObserver();  
        Observer o2 = new GuiSignChangeObserver();  
  
        v.addObserver( o1 );  
        v.addObserver( o2 );  
  
        v.run();  
    }  
}
```

## GuiChangeObserver – Update

```
public class GuiChangeObserver
    implements java.util.Observer {
    public void update( Observable s, Event e ) {
        if( ! (e instanceof ChangeEvent) ) return;
        ChangeEvent ce = (ChangeEvent) e;
        if( ce instanceof IncreaseEvent ) {
            sign.setForeground( Color.RED );
            sign.setText( "++++" );
        } else { ... }
        value.setText( "New Value: " + ce.getValue() );
        change.setText( "Change: " + ce.getChange() );
    }
    ...
}
```

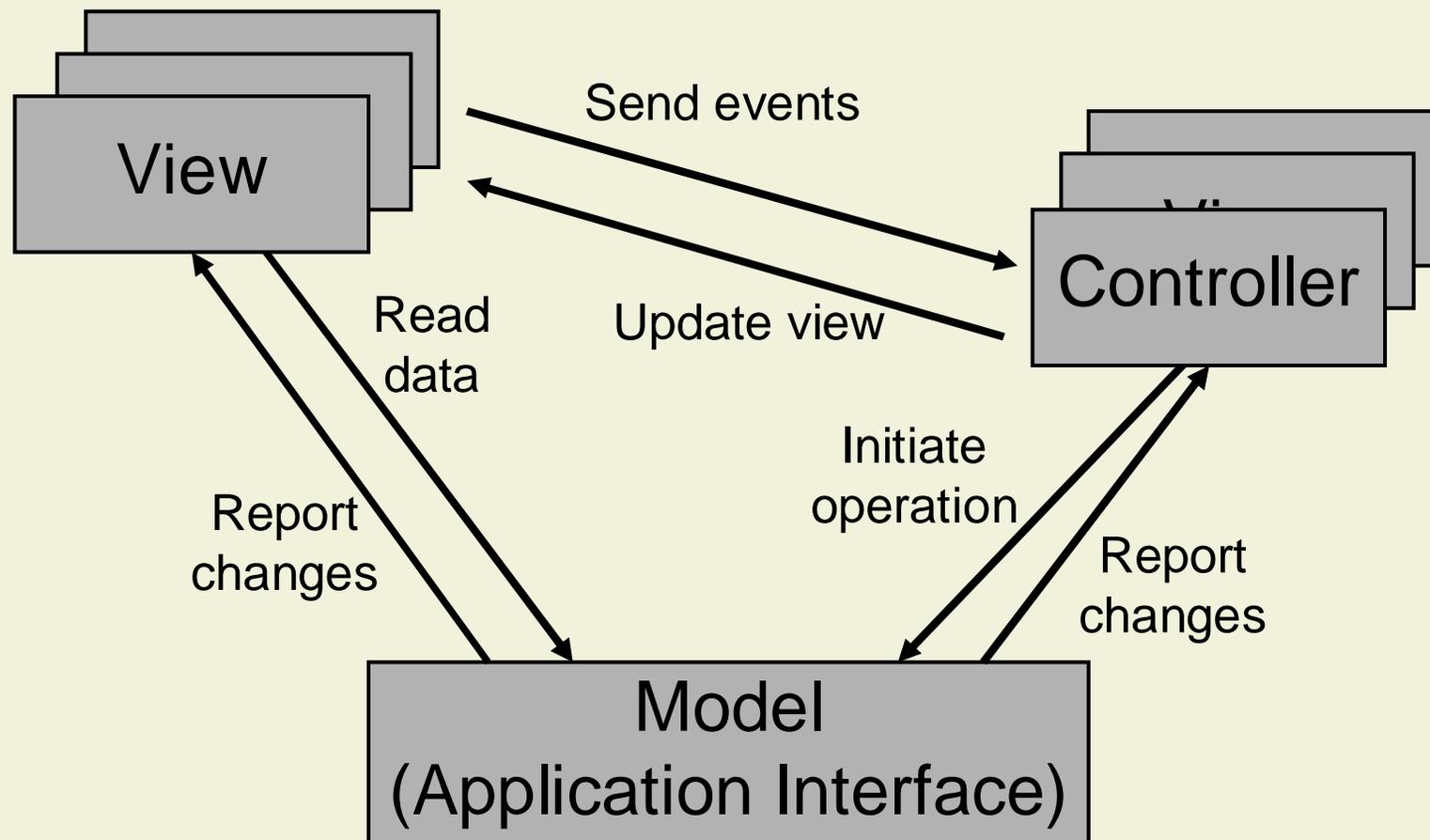
# Adding Components

```
public Component createComponents() {  
    JPanel pane = new JPanel();  
    pane.setBorder(  
        BorderFactory.createEmptyBorder(  
            30, 30, // top, left  
            30, 30) // bottom, right  
        );  
    pane.setLayout(new GridLayout(0, 3));  
    pane.add(sign);  
    pane.add(value);  
    pane.add(change);  
  
    return pane;  
}
```

# Screenshot & Demo



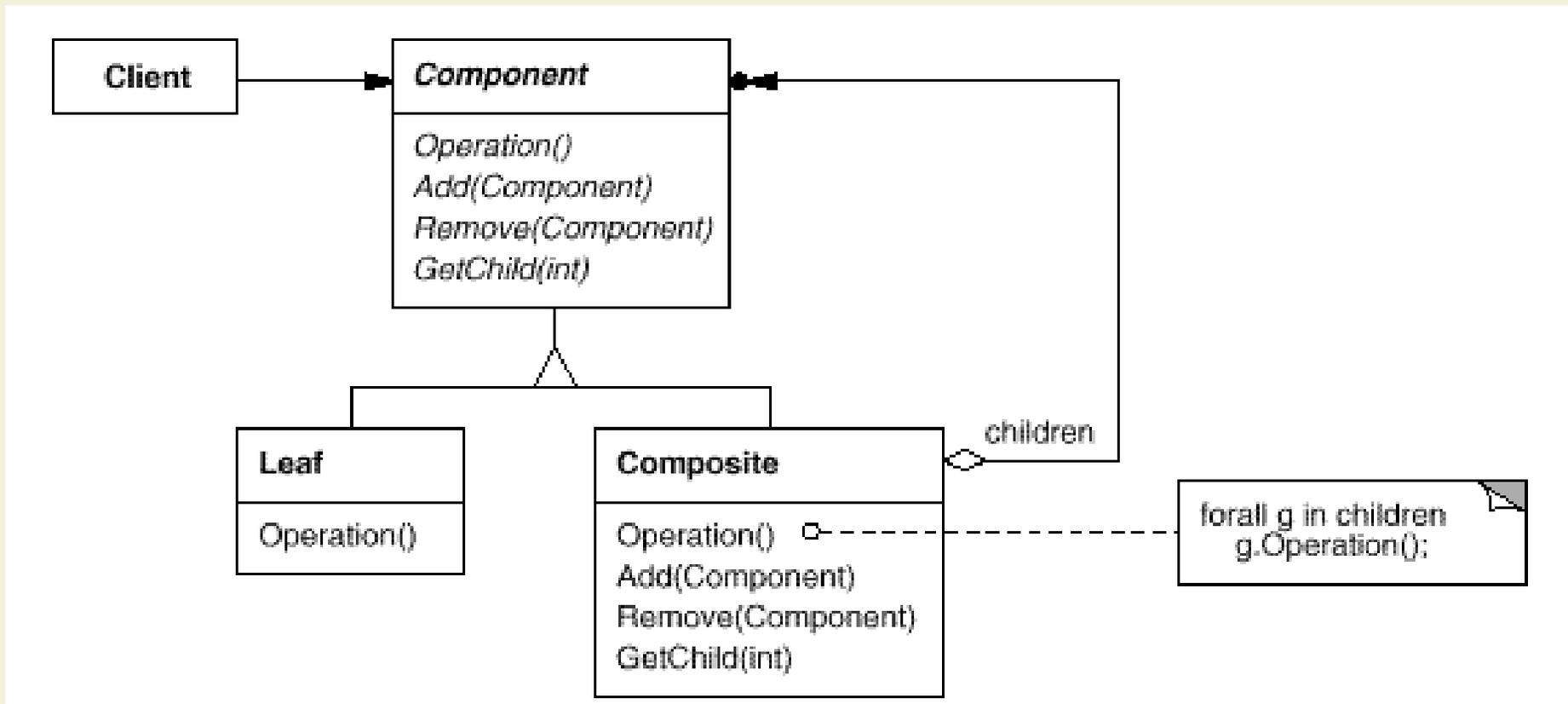
# Model-View-Controller Architecture



## Adding a Simple Reset Controller

```
button = new JButton();  
button.addActionListener(  
    new ActionListener() {  
        public void actionPerformed(  
            ActionEvent e) {  
                theValue.reset();  
            }  
    } );  
  
pane.add(button, BorderLayout.CENTER);
```

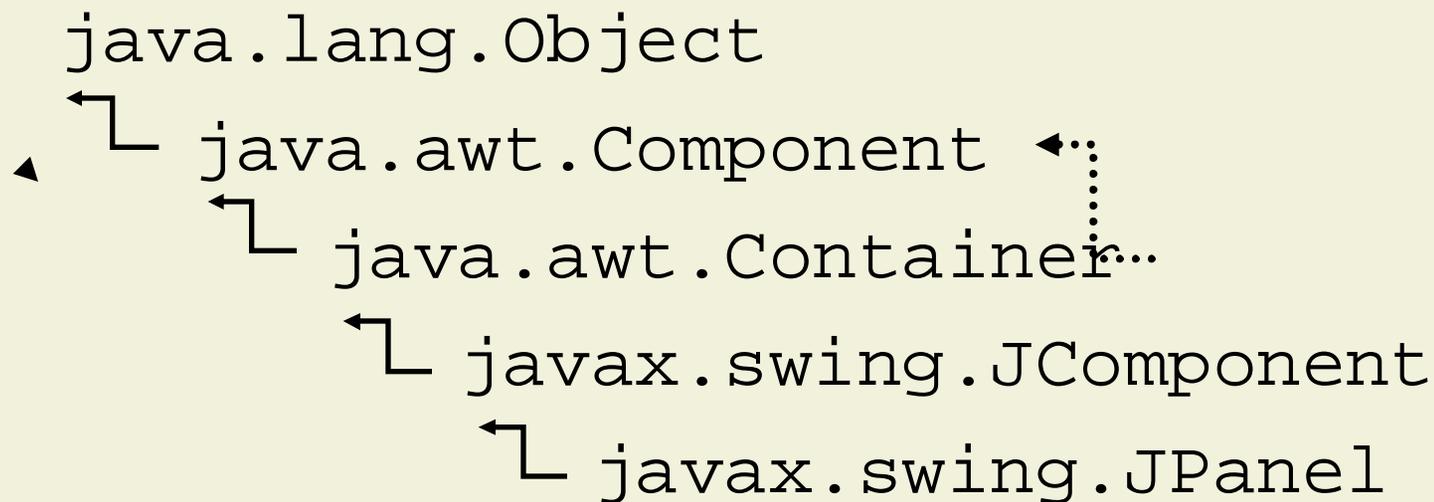
# Composite Pattern



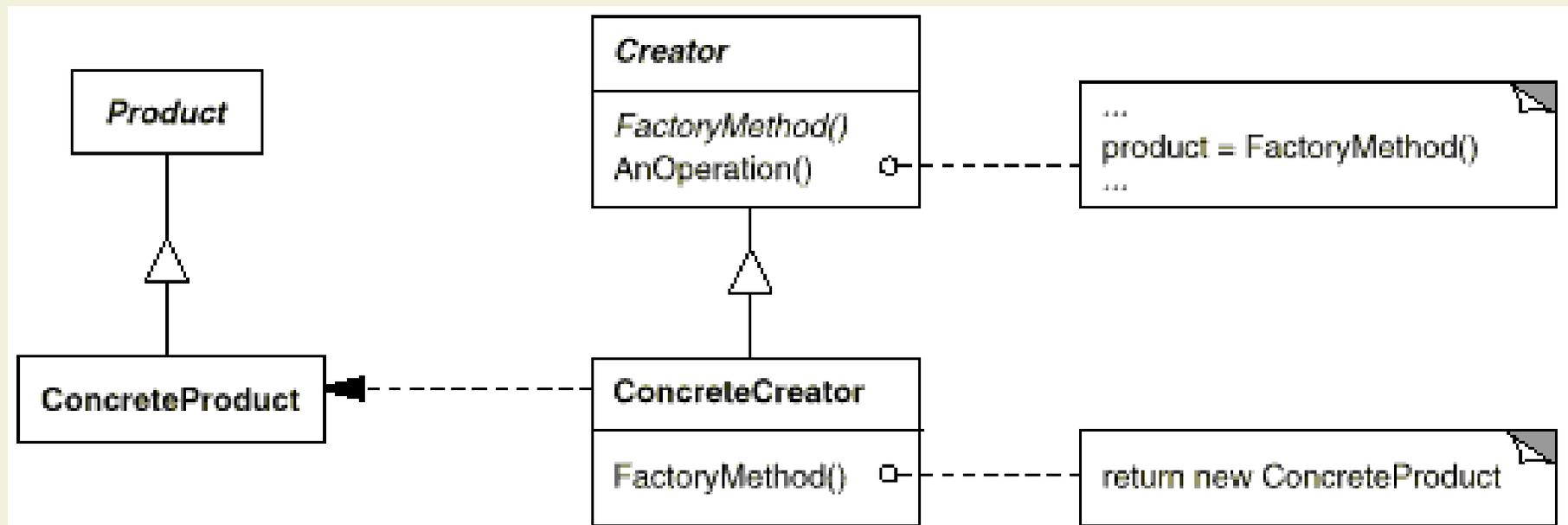
From: Gamma, Helm, Johnson, Vlissides: "Design Patterns"

# Composite Pattern in Swing

- Used many times
- Composite is called Container

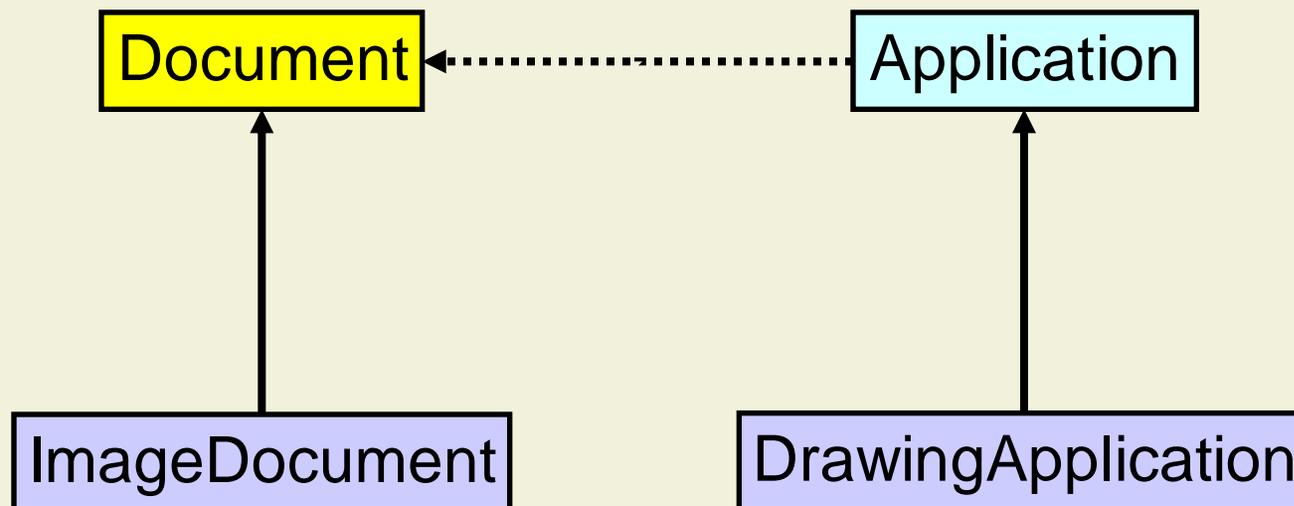


# Homework 2 – Factory Method Pattern



From: Gamma, Helm, Johnson, Vlissides: "Design Patterns"

# Example Classes



# Abstract Class Application

```
public abstract class Application {  
    public Application() {  
        documents = new ArrayList<Document>( );  
    }  
  
    // the factory method!  
    protected abstract Document createDocument( );  
  
    public void newDocument( ) {  
        Document toadd = createDocument( );  
        documents.add( toadd );  
        toadd.open( );  
    }  
  
    . . .  
}
```

# Interface Document

```
public interface Document {  
    void open( ) ;  
    void save( ) ;  
    void reload( ) ;  
    void close( ) ;  
}
```

# Class ImageDocument

```
public class ImageDocument
    implements Document {
    public void open() {
        System.out.println( "ImageDocument.open" );
    }

    public void save() {
        System.out.println( "ImageDocument.save" );
    }

    . . .
}
```

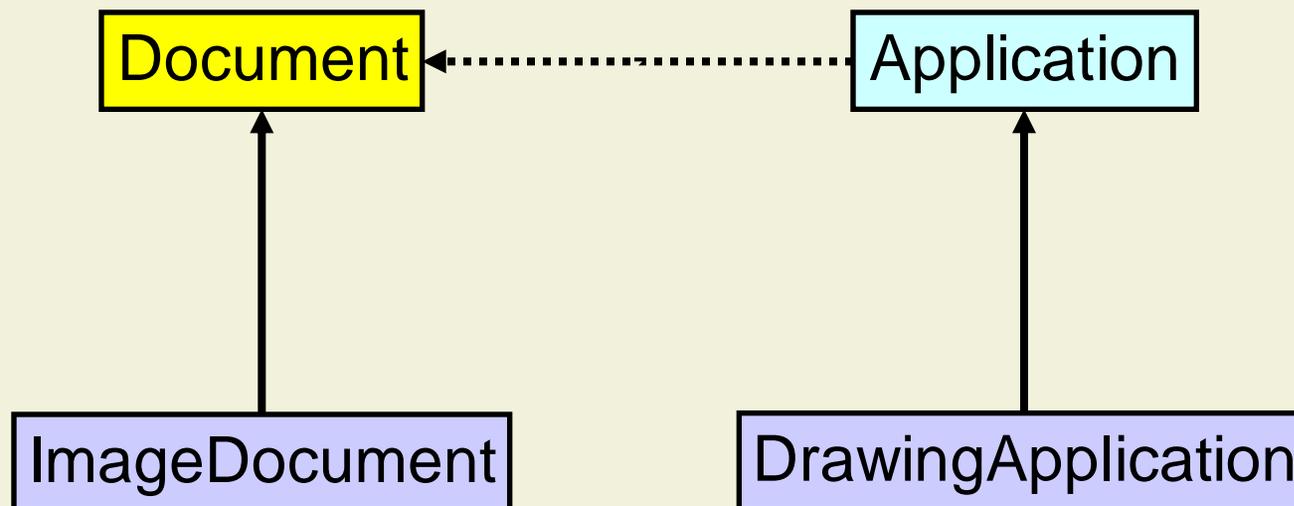
# Class DrawingApplication

```
public class DrawingApplication
    extends Application {
    public DrawingApplication() { super(); }

    // the overridden factory method returns
    // a concrete object of type ImageDocument.
    protected Document createDocument() {
        return new ImageDocument(); }

    public static void main( String[] args ) {
        DrawingApplication appl =
            new DrawingApplication();
        appl.newDocument(); appl.newDocument();
        appl.newDocument(); appl.saveAll();
        appl.closeAll(); }
}
```

# Example Classes



---

# Questions?