

Hand-on 4 Design Patterns

April 2023

Outline

Reminder on previous session: Design of a Drawing App

- 1/ Model-View-Controller pattern
- 2/ Publish&Subscribe pattern
- 3/ core domain classes (Text, Line, Rectangle, Circle, ..)
- 4/ extension classes: Composite, Proxy, Adapter (example: Img)
- 5/ Visitor pattern on domain classes.. Model to View
- 6/ State pattern for mouse click handler
- 7/ Command pattern for undo-redo..

Objective of this Hands-On

Implements the patterns
Obtain a minimalist running application

Easy:
Fully Guided
& Using code snippets

Step 0 : Create/import a project

with maven pom.xml

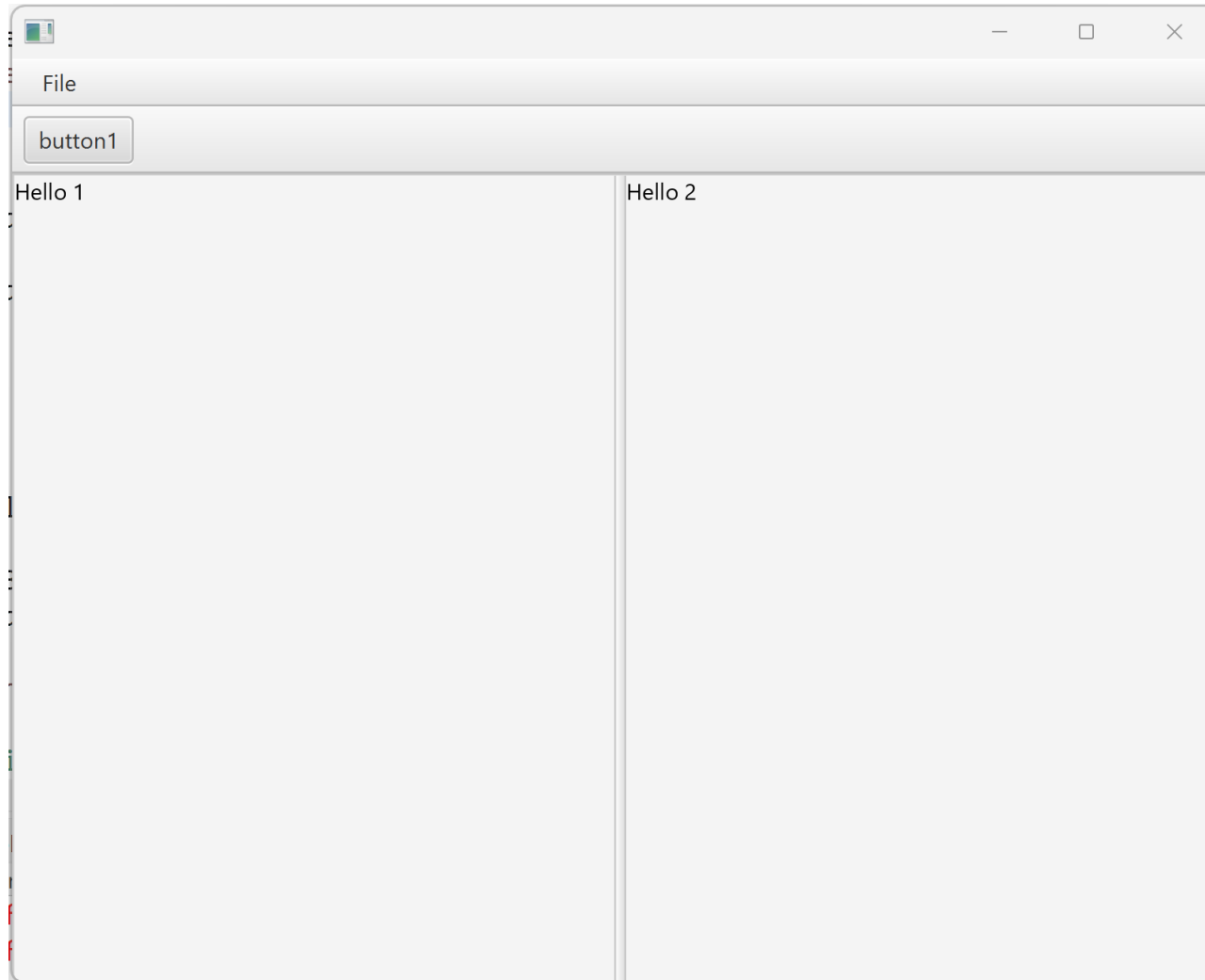
with javafx <dependency>

with a public static void main(String[] args)

with a javafx App


... cf next

Step 0: Run ... Expected GUI Result







Step 0 : clone/download





<https://github.com/Arnaud-Nauwynck/javafx-whiteapp>






[Pull requests](#) [Issues](#) [Codespaces](#) [Marketplace](#) [Explore](#)


 [Arnaud-Nauwynck / javafx-whiteapp](#) Public


 Pin  Unwatch 1  Fork 0  Star 0


[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)


 main  1 branch  0 tags


[Go to file](#) [Add file](#) [Code](#)


 **Arnaud-Nauwynck** added README.md 4e5e045 2 minutes ago 3 commits


 src/main/java/fr/an/tests/javafxwhite... added README.md 2 minutes ago


 .gitignore init javafx whiteapp 16 minutes ago

 LICENSE Initial commit 42 minutes ago

 README.md added README.md 2 minutes ago

 pom.xml init javafx whiteapp 16 minutes ago


 whiteapp-screenshot.png added README.md 2 minutes ago


README.md 


A Minimalist javafx white App


About


a minimalist White Application for javafx

 Readme

 LGPL-2.1 license

 0 stars

 1 watching

 0 forks

Releases

No releases published

[Create a new release](#)

Packages

No packages published

[Publish your first package](#)

README.md

it contains:

javafx dependency in maven pom.xml, mainly:

```
<dependency>
  <groupId>org.openjfx</groupId>
  <artifactId>javafx-controls</artifactId>
  <version>${javafx.version}</version>
</dependency>
```

a java Main to run :

```
import fr.an.tests.javafxwhiteapp.ui.SimpleApp;
import javafx.application.Application;

public class SimpleAppMain {
    public static void main(String[] args) {
        Application.launch(SimpleApp.class, args);
    }
}
```

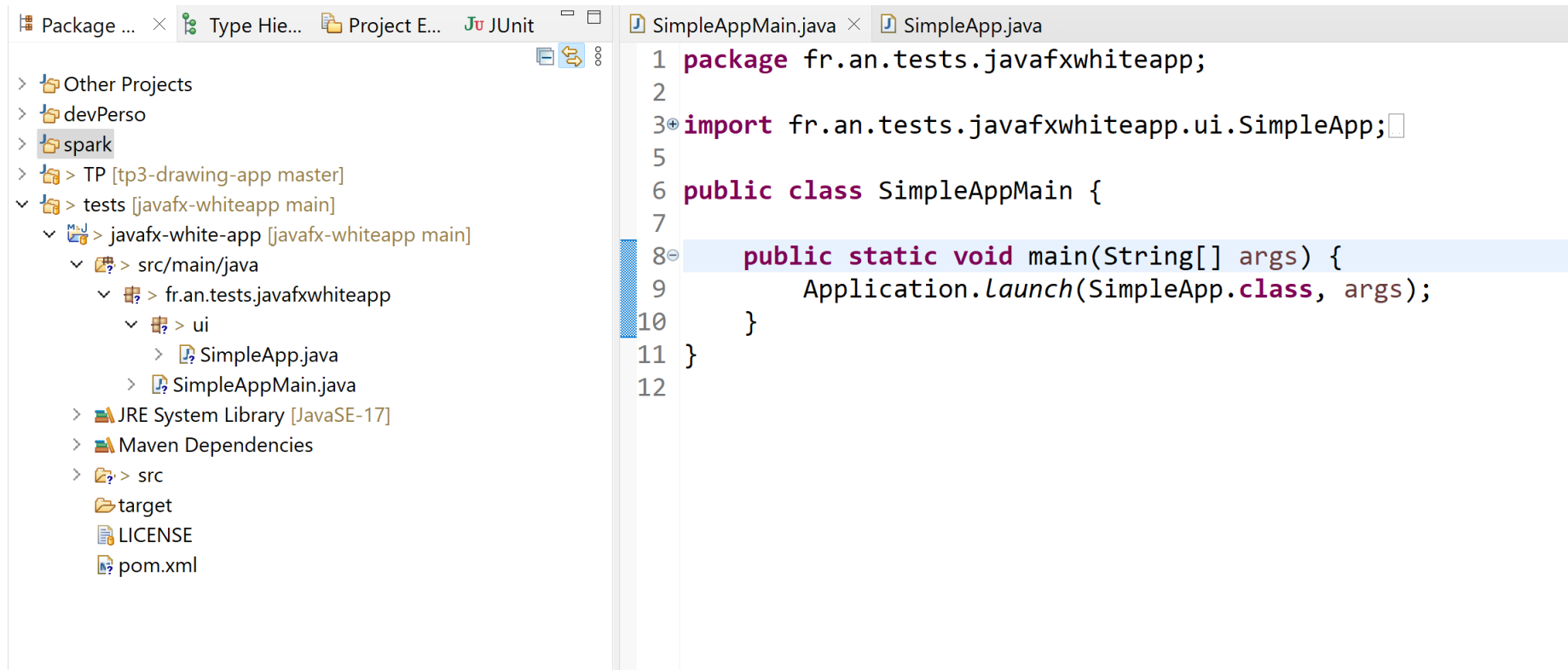
a javafx Application with

```
MenuBar
-> "File" Menu
    -> "Open" MenuItem
    -> "Save" MenuItem
Toolbar
-> a Button
a SplitPane
-> View1 on left
-> View2 on right
```

See code:

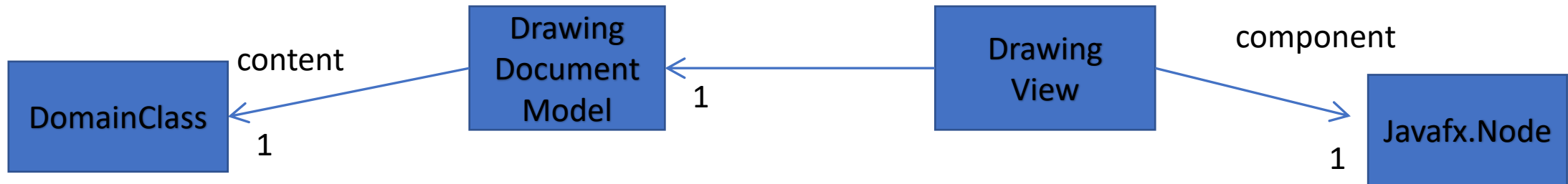
```
import javafx.application.Application;
```

Step 1: Import in Eclipse/Idea + Run It



Step 1 : MVC Design Pattern

Model-View-(Controler)

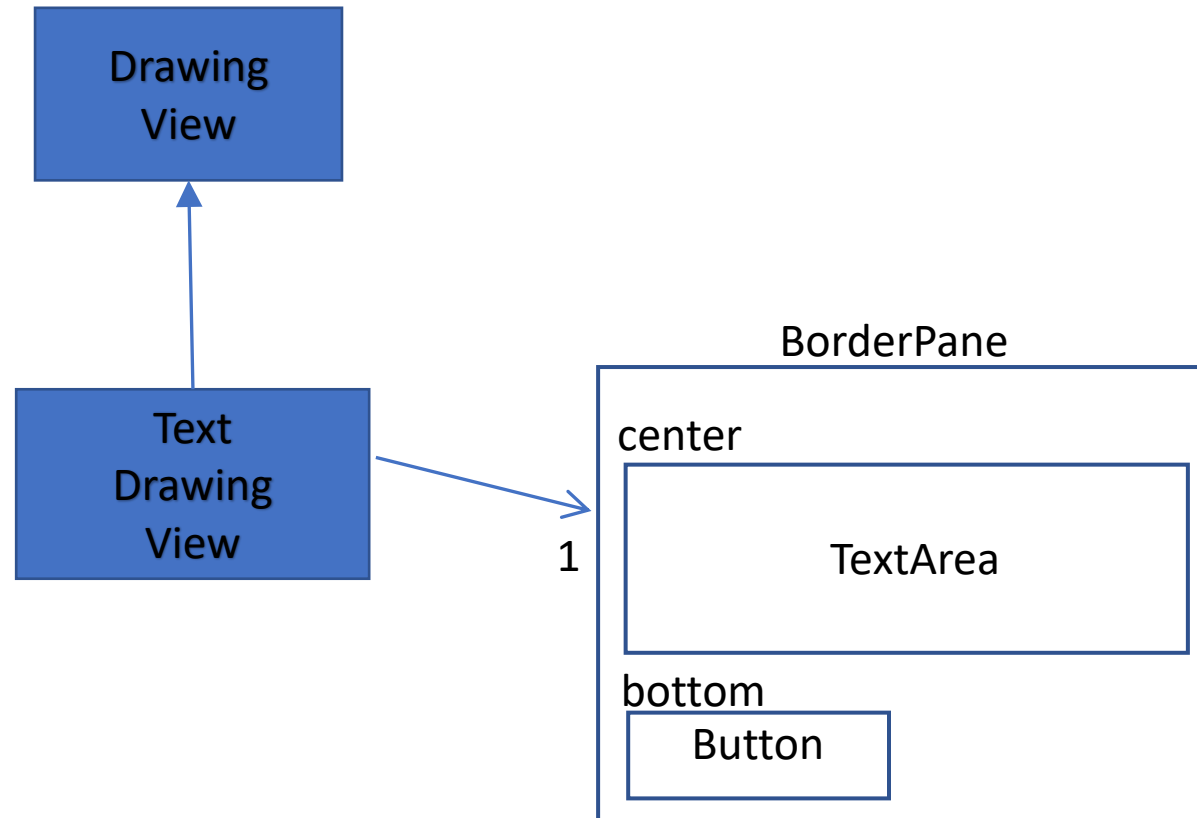


Step 1 : code ..

```
public class DrawingDocModel {  
  
    public String documentName;  
  
    // to be replaced next  
    // by Drawing AST classes  
    public String content;  
  
    // more later: Publisher design-pattern  
  
}
```

```
public abstract class DrawingView {  
  
    protected DrawingDocModel model;  
  
    public DrawingView(DrawingDocModel model) {  
        this.model = model;  
    }  
  
    public abstract javafx.scene.Node getComponent();  
  
    // more later: Subscriber design-pattern  
  
}
```

Step 2: create a simple (text) DrawingView sub-class



Step 2 code

```
public class TextDrawingView extends DrawingView {

    protected BorderPane component;
    protected TextArea textArea;
    protected Button applyButton;

    public TextDrawingView(DrawingDocModel model) {
        super(model);
        this.component = new BorderPane();
        this.textArea = new TextArea();
        component.setCenter(textArea);
        this.applyButton = new Button("Apply");
        component.setBottom(applyButton);
        refreshModelToView();
    }

    @Override
    public Node getComponent() {
        return component;
    }

    public void refreshModelToView() {
        String text = model.getContent();
        textArea.setText(text);
    }
}
```

Step 3: in main app
instanciate 1 model and bind 2 Views

```
model = new Model()
```

```
view1 = new TextDrawingView1(model)
```

```
view2 = new TextDrawingView1(model)
```

```
add views
```

Step 3 code

➡ DrawingDocModel model = new DrawingDocModel();
model.setContent("drawing content will go here later");

```
... ..  
{ // SplitPane( view1 | view2 )  
  VBox view1 = new VBox();  
  view1.getChildren().add(new Text("Hello 1"));  
  
  VBox view2 = new VBox();  
  view2.getChildren().add(new Text("Hello 2"));  
  
  SplitPane splitViewPane = new SplitPane(view1, view2);  
  mainBorderPanel.setCenter(splitViewPane);  
}
```

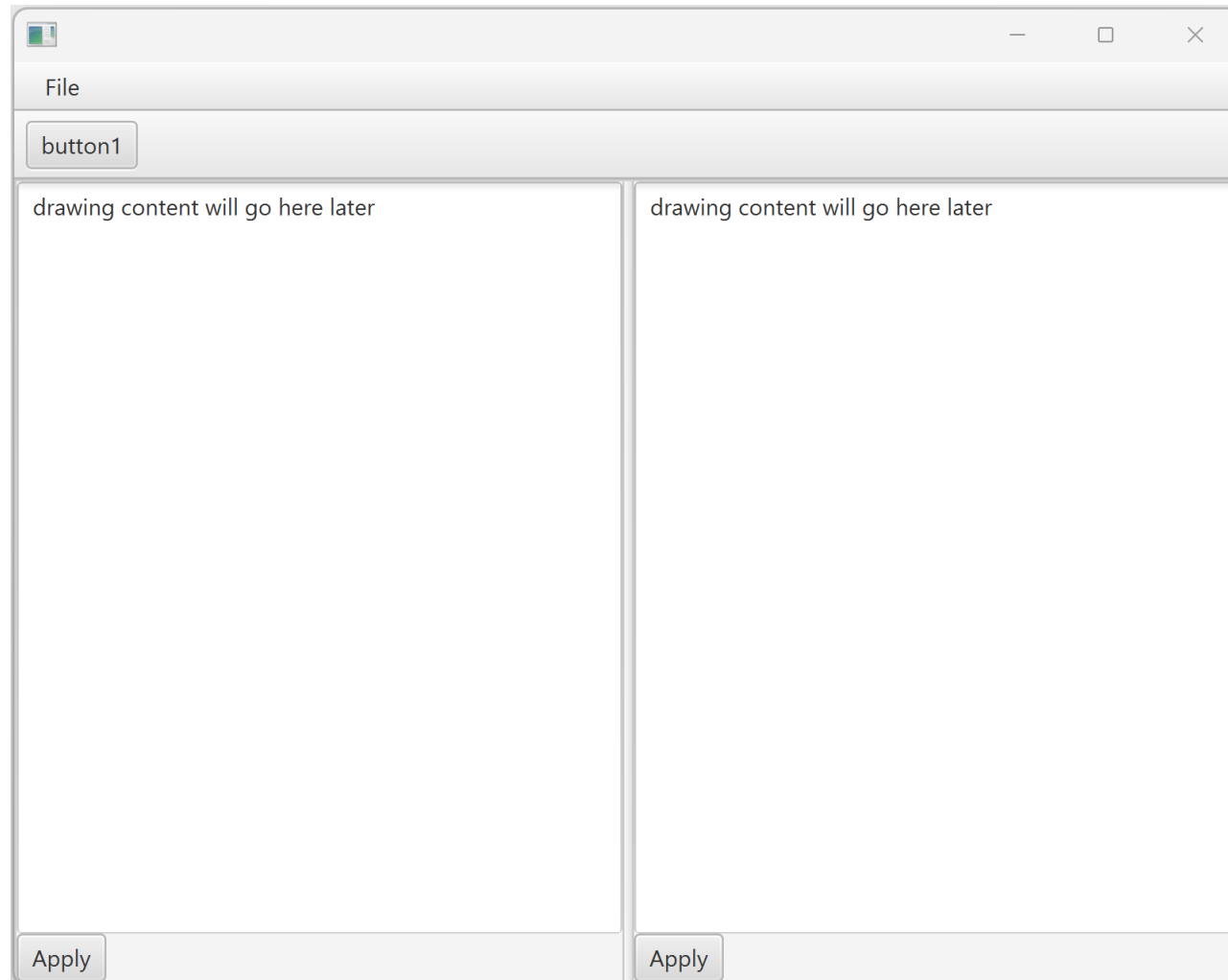
```
Scene scene = new Scene(mainBorderPanel, 640, 480);  
stage.setScene(scene);  
stage.show();  
}
```

{ // SplitPane(view1 | view2)
 TextDrawingView view1 = new TextDrawingView(model);
 Node view1Comp = view1.getComponent();

TextDrawingView view2 = new TextDrawingView(model);
Node view2Comp = view2.getComponent();

SplitPane splitViewPane = new SplitPane(view1Comp, view2Comp);
mainBorderPanel.setCenter(splitViewPane);
}

Step 3 Expected Result



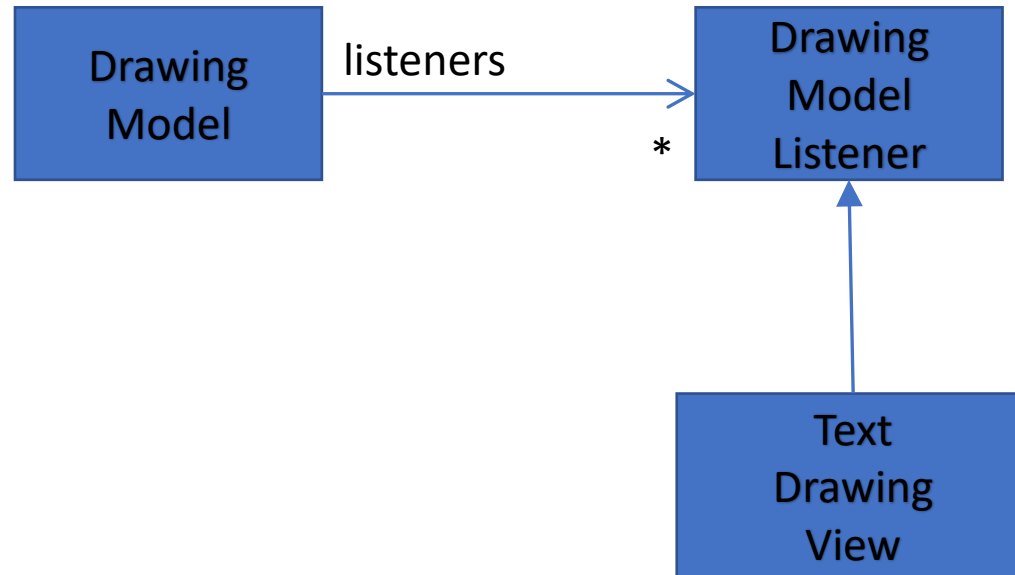
Step 4 : Publish & Subscribe Design Pattern

Declare « interface DrawingModelListener »

Add on model-side

```
« List< DrawingModelListener> listeners = new ArrayList<>(); »  
methods addListener / removeListener / fireListenerChange()
```

Add « implements DrawingModelListener » on view-side + model.addListener(this)



Step 4 code (1/3)

```
protected List<DrawingModelListener> listeners = new ArrayList<>();
```

```
public void setContent(String content) {  
    this.content = content;  
    fireModelChange();  
}
```

```
public void addListener(DrawingModelListener p) {  
    this.listeners.add(p);  
}
```

```
public void removeListener(DrawingModelListener p) {  
    this.listeners.remove(p);  
}
```

```
protected void fireModelChange() {  
    for(DrawingModelListener listener : listeners) {  
        listener.onModelChange();  
    }  
}
```

```
public interface DrawingModelListener {  
    public void onModelChange();  
}
```

Step 4 : Code (2/3)

```
public class TextDrawingView extends DrawingView implements DrawingModelListener {  
  
    public TextDrawingView(DrawingDocModel model) {  
        model.addListener(this); // publish&subscribe design pattern  
    }  
  
    protected void refreshModelToView() {  
        String text = model.getContent();  
        textArea.setText(text);  
    }  
  
    @Override  
    public void onModelChange() {  
        System.out.println("(from subscribe): model to view change");  
        refreshModelToView();  
    }  
}
```

Step 4 : Alternative Code (3/3)

... do not expose « public » Listener

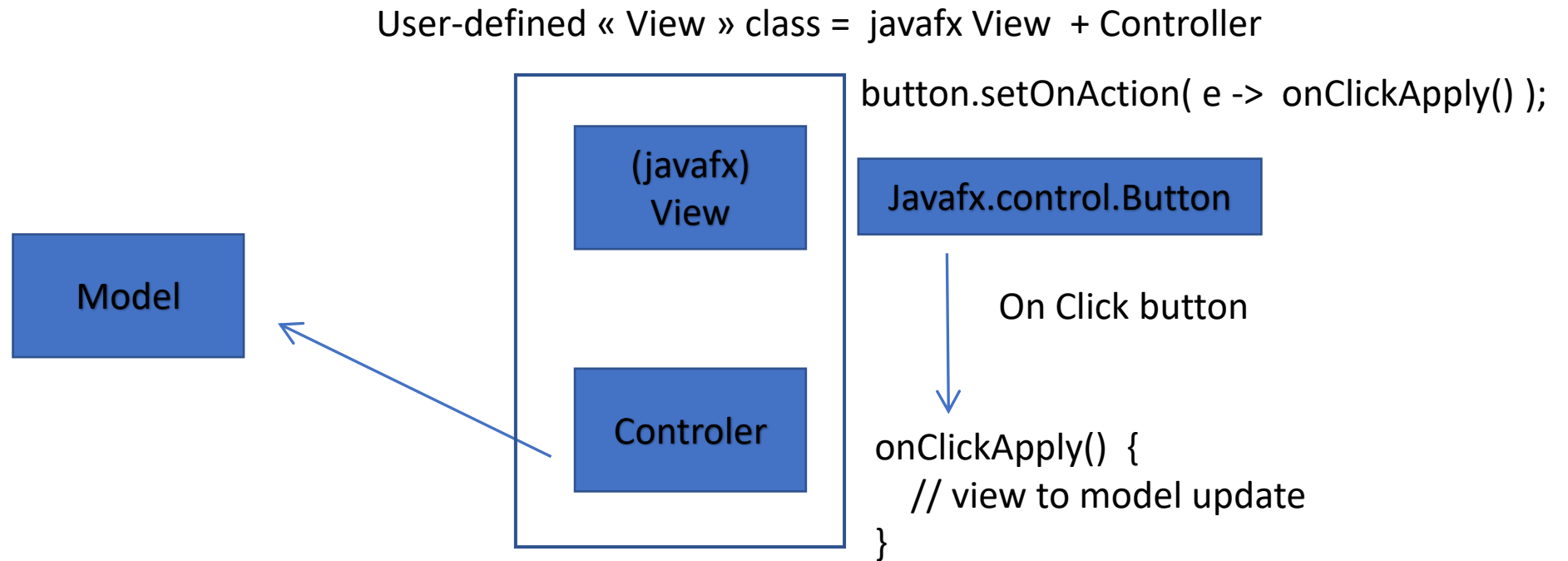
```
public class TextDrawingView extends DrawingView {
```

```
    protected DrawingModelListener innerListener = new DrawingModelListener() {  
        @Override  
        public void onModelChange() {  
            System.out.println("(from subscribe): model to view change");  
            refreshModelToView();  
        }  
    };
```

```
    public TextDrawingView(DrawingDocModel model) {  
        ...  
        model.addListener(this.innerListener); // publish&subscribe design pattern  
    }
```

```
    protected void refreshModelToView() {  
        String text = model.getContent();  
        textArea.setText(text);  
    }
```

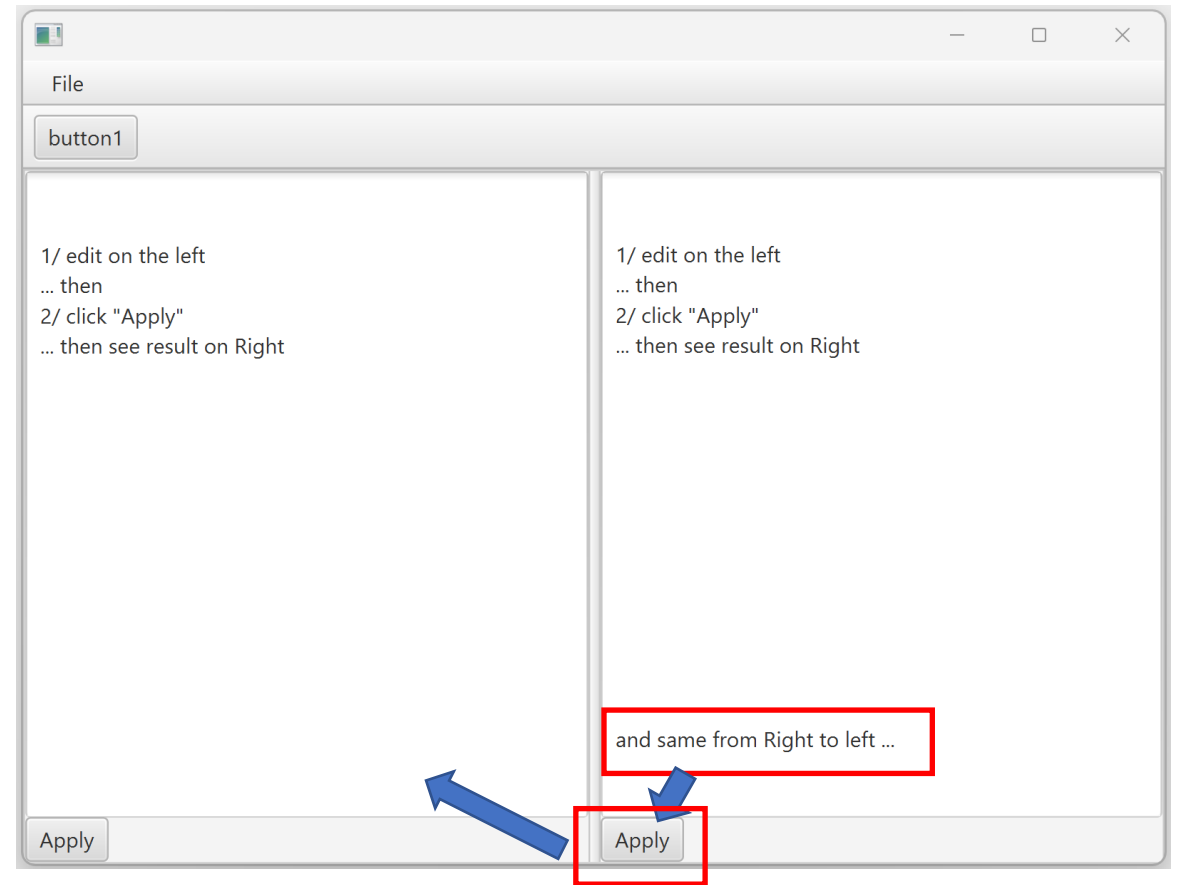
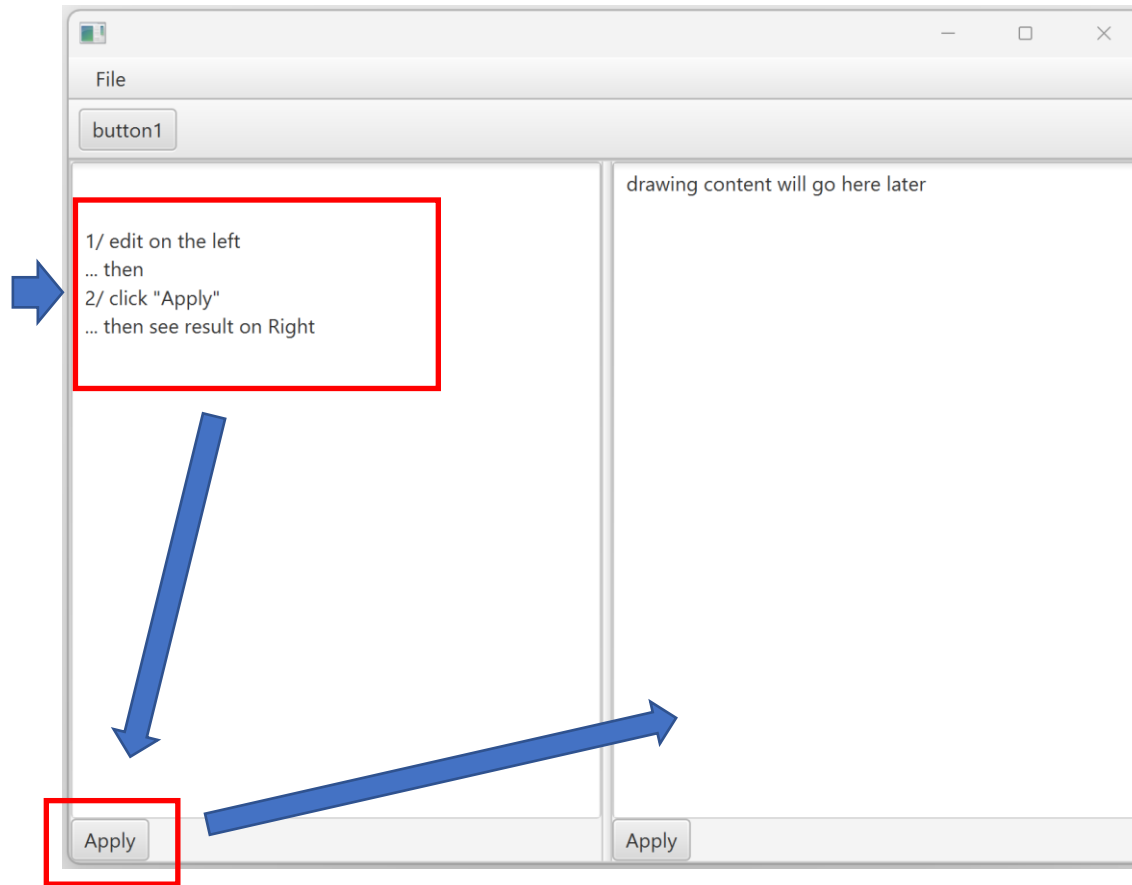
Step 5 : MVC... C=Controller



Step 5: code

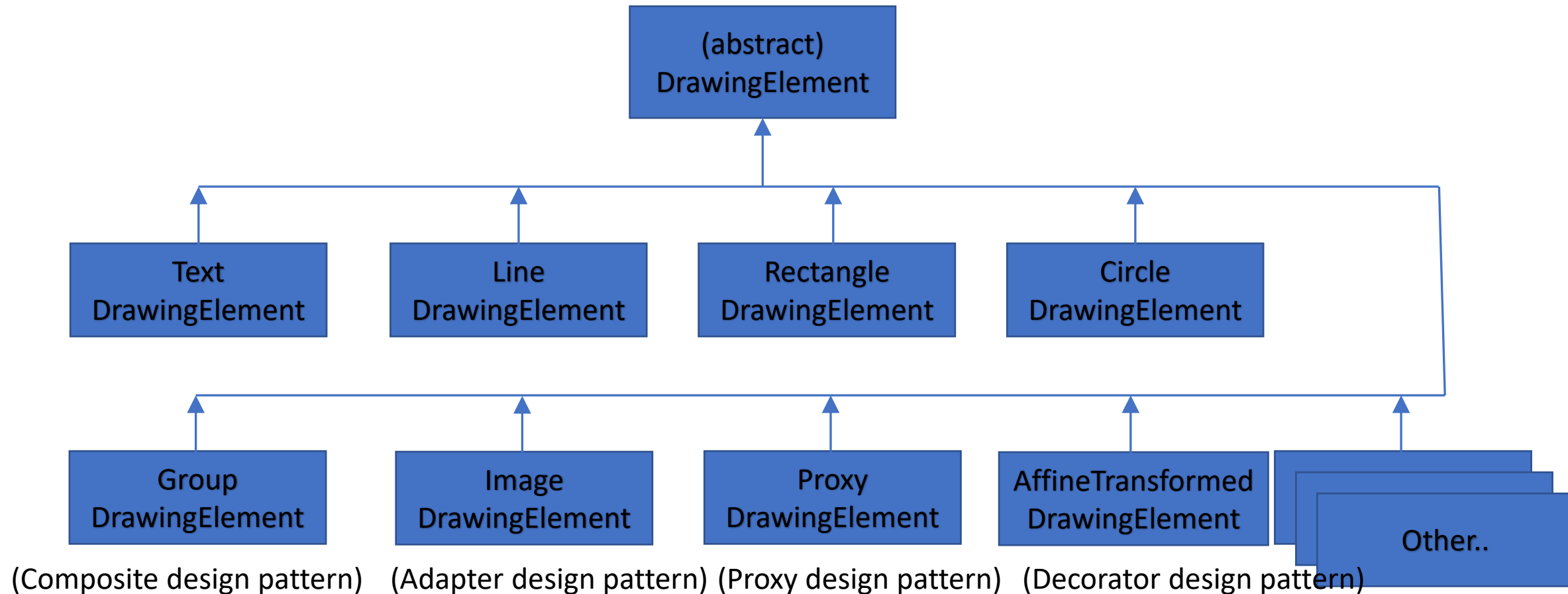
```
public TextDrawingView(DrawingDocModel model) {  
    ...  
    this.applyButton = new Button("Apply");  
    applyButton.setOnAction(e -> onClickApply());  
}  
  
private void onClickApply() {  
    System.out.println("apply view to model update");  
    String text = textArea.getText();  
    model.setContent(text); // => fireModelChange ..  
}
```

Step 5 : Expected Result



Step 6 : core domain classes

design pattern... AST class hierarchy, Interpreter



Step 6: code (1/4)

```
/**
 * abstract base class for Drawing element
 * AST class hierarchy
 * see sub-classes:
 * <ul>
 * <li> TextDrawingElement </li>
 * <li> LineDrawingElement </li>
 * <li> RectanleDrawingElement </li>
 * <li> CircleDrawingElement </li>
 * <li> ImageDrawingElement (adapter design pattern, to image: png/jpg/gif/.. )</li>
 * <li> GroupDrawingElement (composite design-pattern) </li>
 * <li> other.. </li>
 * </ul>
 */
public abstract class DrawingElement {

    // nothing except
    // TOADD: visitor design pattern
}
```


Step 6 : code(2/4)

```
public class BaseDrawingElements {  
  
    public static class TextDrawingElement extends DrawingElement {  
        public String text;  
        public DrawingPt pos;  
        public Map<String,Object> properties; // font, size, color,  
    }  
  
    public static class LineDrawingElement extends DrawingElement {  
        public DrawingPt start;  
        public DrawingPt end;  
        public Map<String,Object> properties; // width, stroke, color, ..  
    }  
  
    public static class RectangleDrawingElement extends DrawingElement {  
        public DrawingPt upLeft;  
        public DrawingPt downRight;  
        public Map<String,Object> properties; // width, stroke, color, ..  
    }  
  
    public static class CircleDrawingElement extends DrawingElement {  
        public DrawingPt center;  
        public double radius;  
        public Map<String,Object> properties; // width, stroke, color, ..  
    }  
}  
  
public class DrawingPt {  
  
    public double x;  
    public double y;  
  
    public DrawingPt() {  
    }  
  
    public DrawingPt(  
        double x,  
        double y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

Step 6 code (3/4)

```
/**
 * Composite design pattern
 */
public static class GroupDrawingElement extends DrawingElement {
    public List<DrawingElement> elements = new ArrayList<>();
}

/**
 * Adapter design pattern, for javafx.scene.image.Image
 */
public static class ImageDrawingElement extends DrawingElement {
    public javafx.scene.image.Image image; // => url, mimeType, data
}
```

Step 6 code (4/4)

```
/**
 * Proxy design pattern
 * example: including part from another document
 */
public static class ProxyDrawingElement extends DrawingElement {
    public DrawingElement underlying;
}

/**
 * Decorator design pattern
 * For geometrical affine transformation
 */
public static class AffineTransformedDrawingElement extends DrawingElement {
    public DrawingElement underlying;
    public DrawingPt translate;
    public double rotateAngle;
    public double scale;
}
```

Step 7 : instantiate a simple Drawing Text + Line + Rectangle + Circle

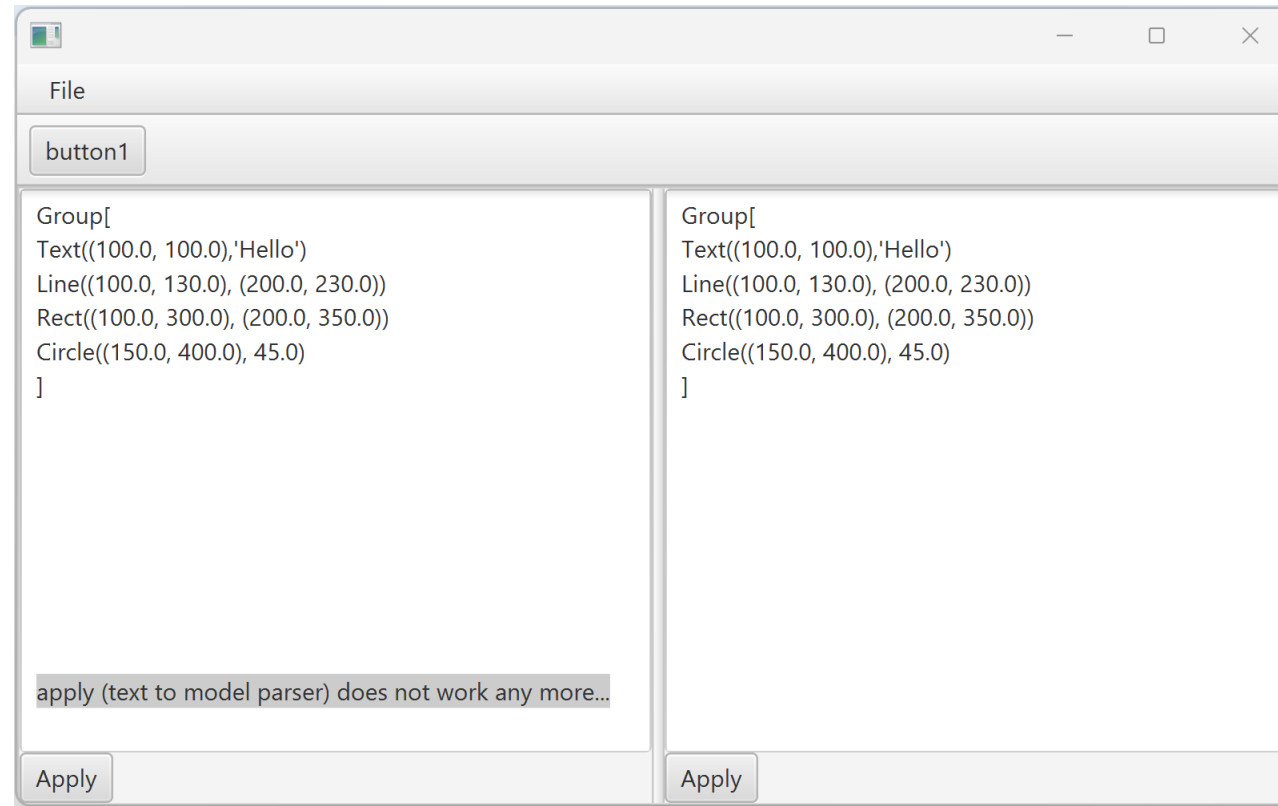
```
public GroupDrawingElement createSimpleDrawing() {  
    TextDrawingElement text = new TextDrawingElement("Hello", new DrawingPt(100, 100));  
  
    LineDrawingElement line = new LineDrawingElement(new DrawingPt(100, 130),  
                                                       new DrawingPt(200, 230));  
  
    RectangleDrawingElement rectangle = new RectangleDrawingElement(  
                                         new DrawingPt(100, 300), new DrawingPt(200, 350));  
  
    CircleDrawingElement circle = new CircleDrawingElement(new DrawingPt(150, 400), 45);  
  
    GroupDrawingElement res = new GroupDrawingElement();  
    res.addAll(text, line, rectangle, circle);  
    return res;  
}
```

Step 8 : change Model-> to use DrawingElement
+ naive code for model to text
(cf next ... using Visitor)

```
public class DrawingDocModel {  
    protected String documentName;  
    protected DrawingElement content; // was String before step8
```

```
// in main:  
DrawingDocModel model = new DrawingDocModel();  
DrawingElement content = createSimpleDrawing();  
model.setContent(content);
```

Step 8 : expected Result



Step 8 : code (1/3)

```
protected void refreshModelToView() {
    DrawingElement content = model.getContent();
    String text = recursiveElementToText(content);
    textArea.setText(text);
}

private String recursiveElementToText(DrawingElement drawingElement) {
    if (drawingElement instanceof TextDrawingElement) {
        TextDrawingElement p = (TextDrawingElement) drawingElement;
        return "Text(" + p.pos + ", '" + p.text + "')";
    } else if (drawingElement instanceof LineDrawingElement) {
        LineDrawingElement p = (LineDrawingElement) drawingElement;
        return "Line(" + p.start + ", " + p.end + ")";
    } // } else .. Cf next page
```

Step 8 code (2/3)

```
} else if (drawingElement instanceof RectangleDrawingElement) {
    RectangleDrawingElement p = (RectangleDrawingElement) drawingElement;
    return "Rect(" + p.upLeft + ", " + p.downRight + ")";
} else if (drawingElement instanceof CircleDrawingElement) {
    CircleDrawingElement p = (CircleDrawingElement) drawingElement;
    return "Circle(" + p.center + ", " + p.radius + ")";
} else if (drawingElement instanceof GroupDrawingElement) {
    GroupDrawingElement p = (GroupDrawingElement) drawingElement;
    StringBuilder sb = new StringBuilder();
    sb.append("Group[\n");
    for(DrawingElement child: p.elements) {
        // *** recurse ***
        sb.append(recursiveElementToText(child));
        sb.append("\n");
    }
    sb.append("]");
    return sb.toString();
} else {
    return "not implemented/recognized drawingElement "+
        drawingElement.getClass().getName();
}
}
```


Step 9: introduce the Visitor design-pattern

instead of ugly « if (instanceof ..) downcast ..else »

Step 9 : code (1/2)

```
public abstract class DrawingElementVisitor {

    public abstract void caseText(TextDrawingElement p);
    public abstract void caseLine(LineDrawingElement p);
    public abstract void caseRect(RectangleDrawingElement p);
    public abstract void caseCircle(CircleDrawingElement p);
    public abstract void caseGroup(GroupDrawingElement p);

    public abstract void caseOther(DrawingElement p);
}

    public abstract class DrawingElement {
        /**
         * Visitor design pattern
         * implement in sub-class, to call <code> visitor.caseXX(this); </code>
         */
        public abstract void accept(DrawingElementVisitor visitor);
    }
```

Step 9 : code (2/2)

```
class TextDrawingElement ...  
    @Override  
    public void accept(DrawingElementVisitor visitor) {  
        visitor.caseText(this);  
    }  
}
```

```
class LineDrawingElement ...  
    @Override  
    public void accept(DrawingElementVisitor visitor) {  
        visitor.caseLine(this);  
    }  
}
```

```
... class <<XX>>DrawingElement ...  
    @Override  
    public void accept(DrawingElementVisitor visitor) {  
        visitor.case<<XX>>(this);  
    }
```

Step 10 : refactor

«String recursiveElementToText(DrawingElement e) »
to use Visitor design-pattern

Step 10 : code (1/2)

```
private String recursiveElementToText(DrawingElement drawingElement) {  
    TextDrawingElementVisitor visitor = new TextDrawingElementVisitor();  
    drawingElement.accept(visitor);  
    return visitor.result;  
}
```

```
protected static class TextDrawingElementVisitor extends DrawingElementVisitor {  
  
    String result;  
  
    @Override  
    public void caseText(TextDrawingElement p) {  
        result = "Text(" + p.pos + ", '" + p.text + "' )";  
    }  
  
    @Override  
    public void caseLine(LineDrawingElement p) {  
        result = "Line(" + p.start + ", " + p.end + ")";  
    }  
}
```

Step 10 : code (2/2)

```
@Override
public void caseRect(RectangleDrawingElement p) {
    result = "Rect(" + p.upLeft + ", " + p.downRight + ")";
}

@Override
public void caseCircle(CircleDrawingElement p) {
    result = "Circle(" + p.center + ", " + p.radius + ")";
}

@Override
public void caseGroup(GroupDrawingElement p) {
    StringBuilder sb = new StringBuilder();
    sb.append("Group[\n");
    for(DrawingElement child: p.elements) {
        // *** recurse ***
        child.accept(this);
        sb.append(result);
        sb.append("\n");
    }
    sb.append("]");
    result = sb.toString();
}

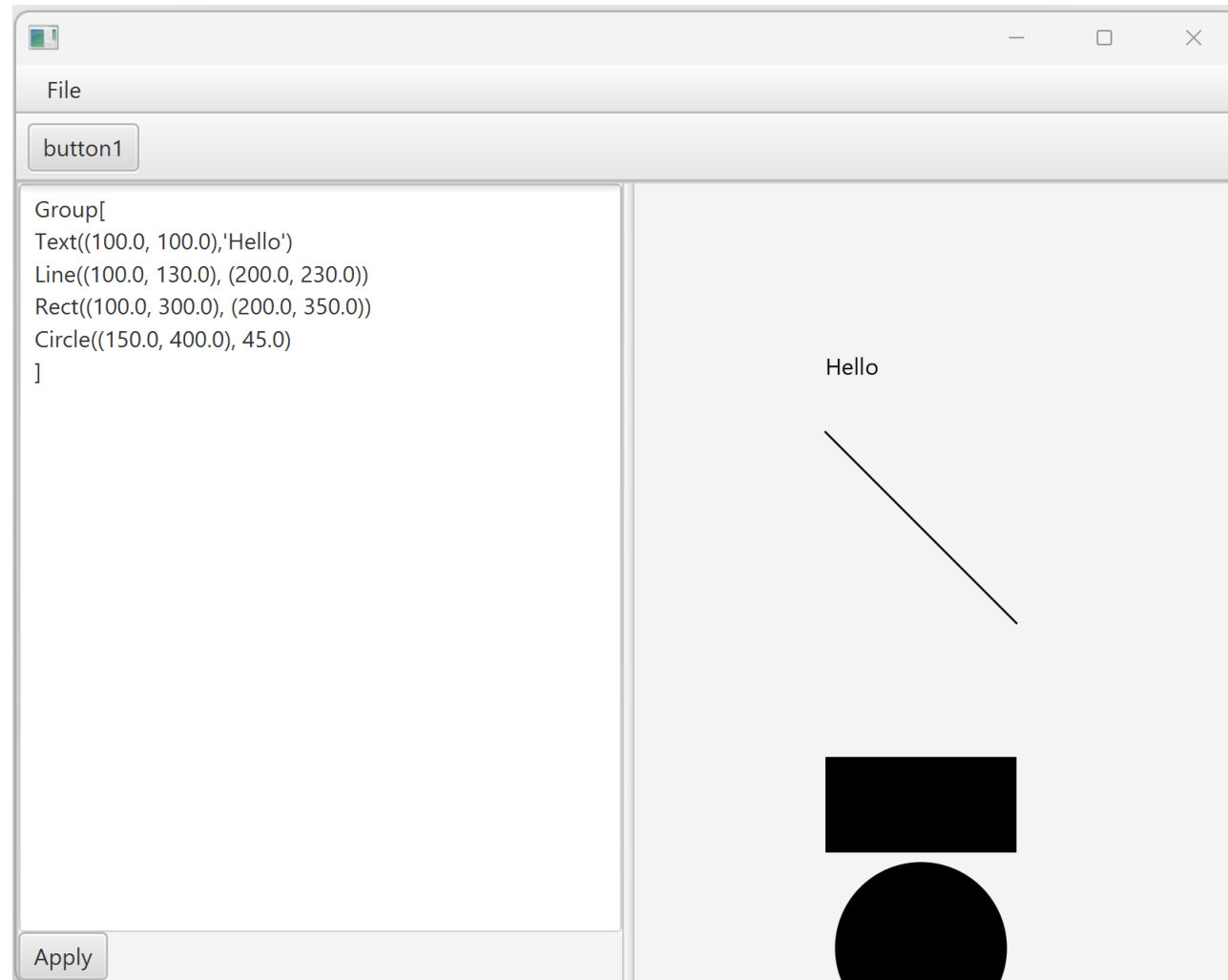
@Override
public void caseOther(DrawingElement p) {
    result = "not implemented/recognized
drawingElement " + p.getClass().getName();
}
```

Step 11 : implement a graphical
DrawingElementView sub-class
(copy&paste from TextDrawingView)

using javafx shape objects

use in application right View
(keep left as Text)

Step 11 : expected Result



Step 11 : code (1/3)

```
{ // SplitPane( view1 | view2 )
  TextDrawingView view1 = new TextDrawingView(model);
  Node view1Comp = view1.getComponent();

  CanvasDrawingView view2 = new CanvasDrawingView(model);
  Node view2Comp = view2.getComponent();

  SplitPane splitViewPane = new SplitPane(view1Comp, view2Comp);
  mainBorderPanel.setCenter(splitViewPane);
}
```

```
public class CanvasDrawingView extends DrawingView {

    protected BorderPane component;
    // to add javafx.scene.shape.* objects converted from model
    protected Pane drawingPane;
```

Step 11 : code (2/3)

```
protected void refreshModelToView() {
    DrawingElement content = model.getContent();
    drawingPane.getChildren().clear();
    JavafxDrawingElementVisitor visitor = new JavafxDrawingElementVisitor();
    content.accept(visitor);
}

protected class JavafxDrawingElementVisitor extends DrawingElementVisitor {
    protected void add(Node node) {
        drawingPane.getChildren().add(node);
    }
    @Override
    public void caseText(TextDrawingElement p) {
        add(new javafx.scene.shape.Text(p.pos.x, p.pos.y, p.text));
    }
    @Override
    public void caseLine(LineDrawingElement p) {
        add(new new javafx.scene.shape.Line(p.start.x, p.start.y, p.end.x, p.end.y));
    }
}
```

Step 11 : code (3/3)

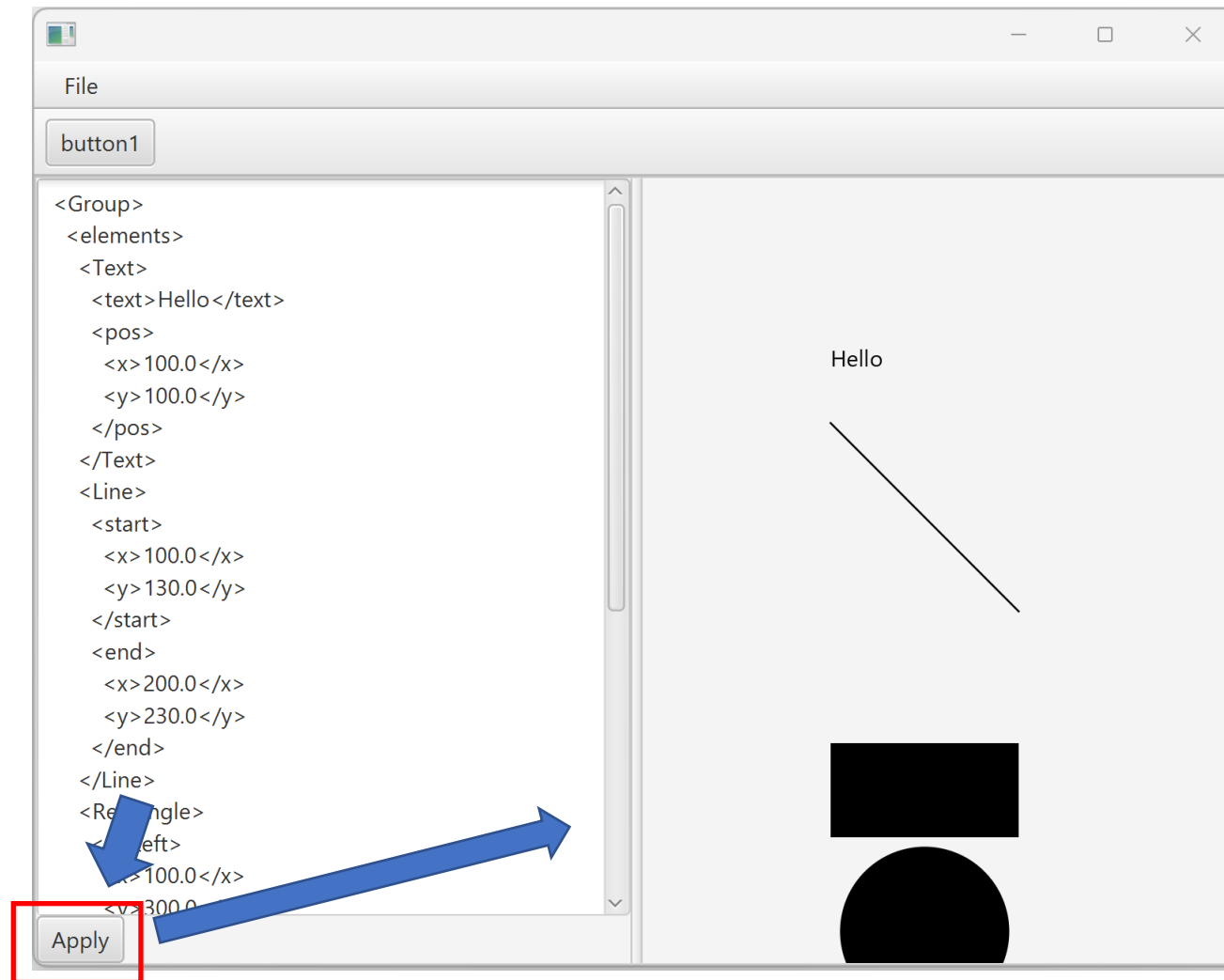
```
@Override
public void caseRect(RectangleDrawingElement p) {
    add(new javafx.scene.shape.Rectangle(p.upLeft.x, p.upLeft.y,
        p.downRight.x-p.upLeft.x, p.downRight.y-p.upLeft.y));
}
@Override
public void caseCircle(CircleDrawingElement p) {
    add(new javafx.scene.shape.Circle(p.center.x, p.center.y, p.radius);
}
@Override
public void caseGroup(GroupDrawingElement p) {
    for(DrawingElement child: p.elements) {
        // *** recurse ***
        child.accept(this);
    }
}
@Override
public void caseOther(DrawingElement p) {
    // "not implemented/recognized drawingElement "+ p.getClass().getName();
}
```

Step 12 : recover Full Edit capabilities

... using XStream library

for formatting model -> text (xml)
and parsing text (xml) -> model

Step 12 : expected result



Step 12 : code (1/3)

```
<dependency>  
  <groupId>com.thoughtworks.xstream</groupId>  
  <artifactId>xstream</artifactId>  
  <version>1.4.20</version>  
</dependency>
```

Step 12 : code (2/3)

```
XStream xstream = createXStream();
```

```
static XStream createXStream() {  
    XStream xstream = new XStream();  
    xstream.addPermission(AnyTypePermission.ANY);  
  
    xstream.alias("Pt", DrawingPt.class);  
    xstream.alias("Text", TextDrawingElement.class);  
    xstream.alias("Line", LineDrawingElement.class);  
    xstream.alias("Circle", CircleDrawingElement.class);  
    xstream.alias("Rectangle", RectangleDrawingElement.class);  
    xstream.alias("Group", GroupDrawingElement.class);  
    return xstream;  
}
```

Step 12 : code (3/3)

```
protected void refreshModelToView() {
    DrawingElement content = model.getContent();
    String text = xstream.toXML(content);
    textArea.setText(text);
}

private void onClickApply() {
    System.out.println("apply view to model update");
    String text = textArea.getText();
    DrawingElement content = (DrawingElement) xstream.fromXML(text);
    model.setContent(content); // => fireModelChange ..
}
```


Step 13 : Add Button Toolbar for editing new « Line »

```
{ // button Toolbar
  Toolbar toolBar = new Toolbar();
  component.setTop(toolBar);

  Button resetToolButton = new Button("Reset");
  resetToolButton.setOnAction(e -> onClickToolReset());
  toolBar.getItems().add(resetToolButton);

  Button newLineButton = new Button("+Line");
  newLineButton.setOnAction(e -> onClickToolNewLine());
  toolBar.getItems().add(newLineButton);
}
```



Step 14 : Tool State Handler

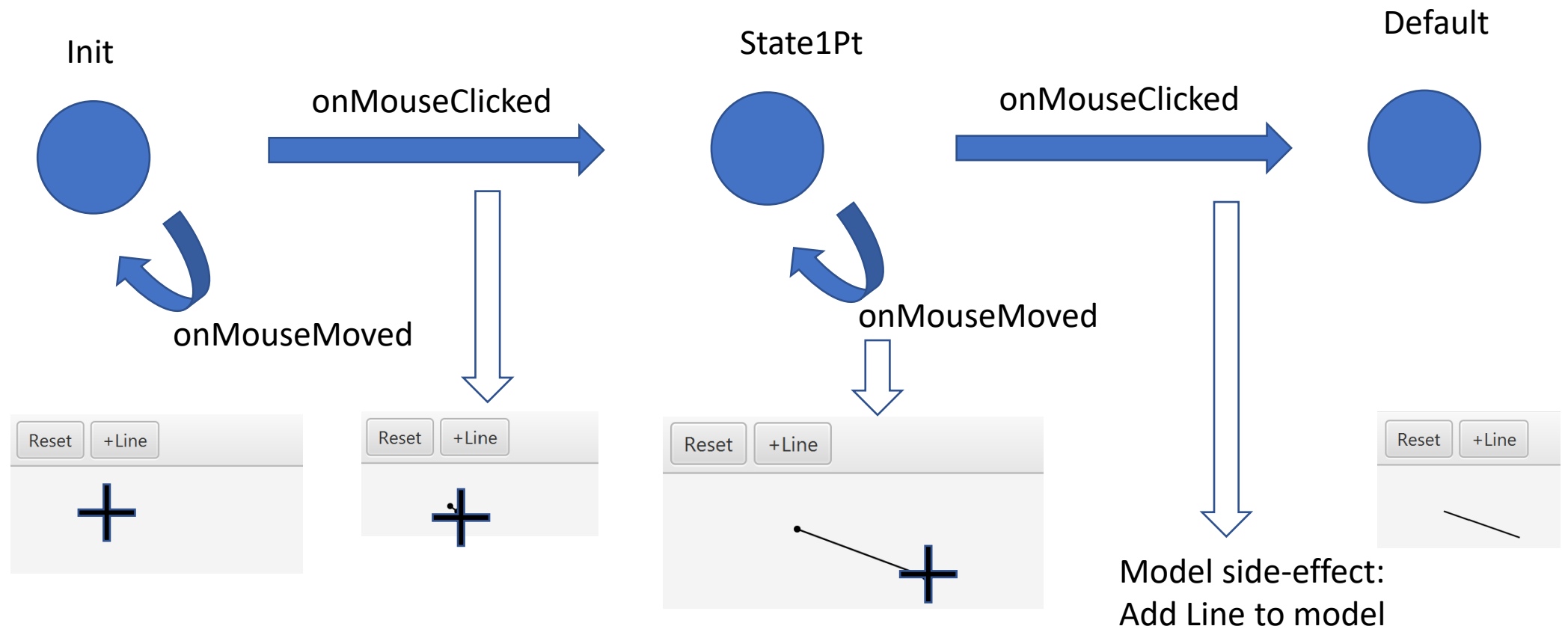
```
public abstract class ToolStateHandler {  
  
    public abstract void onMouseEntered();  
    public abstract void onMouseMove(MouseEvent event);  
    public abstract void onMouseClick(MouseEvent event);  
  
}
```

```
drawingPane.setOnMouseEntered(e -> currToolStateHandler.onMouseEnter());  
drawingPane.setOnMouseMoved(e -> currToolStateHandler.onMouseMoved(e));  
drawingPane.setOnMouseClicked(e -> currToolStateHandler.onMouseClicked(e));
```

Step 14

```
protected class DefaultSelectToolStateHandler extends DefaultToolStateHandler {  
  
    @Override  
    public void onMouseEnter() {  
        drawingPane.setCursor(Cursor.DEFAULT);  
    }  
  
}
```

Step 15 : State Automaton for « new Line »



Step 15: code

```
protected ToolStateHandler currToolStateHandler = new DefaultSelectToolStateHandler();

protected ObservableList<Node> currToolShapes = FXCollections.observableArrayList();
protected CircleDrawingElement currEditLineStartPt;
protected CircleDrawingElement currEditLineEndPt;
protected LineDrawingElement currEditLine;

protected void updateCurrEditTool() {
    drawingPane.getChildren().removeAll(currToolShapes);
    currToolShapes.clear();
    JavafxDrawingElementVisitor visitor = new JavafxDrawingElementVisitor(currToolShapes);
    if (currEditLineStartPt != null) { currEditLineStartPt.accept(visitor); }
    if (currEditLineEndPt != null) { currEditLineEndPt.accept(visitor); }
    if (currEditLine != null) { currEditLine.accept(visitor); }
    drawingPane.getChildren().addAll(currToolShapes);
}
```

Step 15

```
private void onClickToolReset() {
    this.currToolStateHandler = new DefaultToolStateHandler();
    currEditLineStartPt = null;
    currEditLineEndPt = null;
    currEditLine = null;
    refreshModelToView();
}
private void onClickToolNewLine() {
    this.currToolStateHandler = new StateInit_LineToolStateHandler();
}

protected void setToolHandler(ToolStateHandler p) {
    currToolStateHandler = p;
    updateCurrEditTool();
}
```

Step 15

```
protected class StateInit_LineToolStateHandler extends DefaultToolStateHandler {

@Override
public void onMouseEnter() {
    drawingPane.setCursor(Cursor.CROSSHAIR);
}

@Override
public void onMouseClicked(MouseEvent event) {
    double x = event.getX(), y = event.getY();
    DrawingPt pt = new DrawingPt(x, y);
    currEditLineStartPt = new CircleDrawingElement(pt, 2);
    currEditLineEndPt = new CircleDrawingElement(pt, 2);
    currEditLine = new LineDrawingElement(pt, pt);
    updateCurrEditTool();
    setToolHandler(new StatePt1_LineToolStateHandler());
}
```

Step 15

```
protected class StatePt1_LineToolStateHandler extends DefaultToolStateHandler {
    @Override
    public void onMouseEnter() {
        drawingPane.setCursor(Cursor.CROSSHAIR);
    }
    @Override
    public void onMouseMoved(MouseEvent event) {
        double x = event.getX(), y = event.getY();
        currEditLineEndPt.center = currEditLine.end = new DrawingPt(x, y);
        updateCurrEditTool();
    }
    @Override
    public void onMouseClicked(MouseEvent event) {
        LineDrawingElement addToModel = currEditLine;
        GroupDrawingElement content = (GroupDrawingElement) model.getContent();
        content.elements.add(addToModel);
        model.setContent(content);
        currEditLine = null; currEditLineStartPt = null; currEditLineEndPt = null;
        updateCurrEditTool();
        setToolHandler(new DefaultSelectToolStateHandler());
    }
}
```


• •

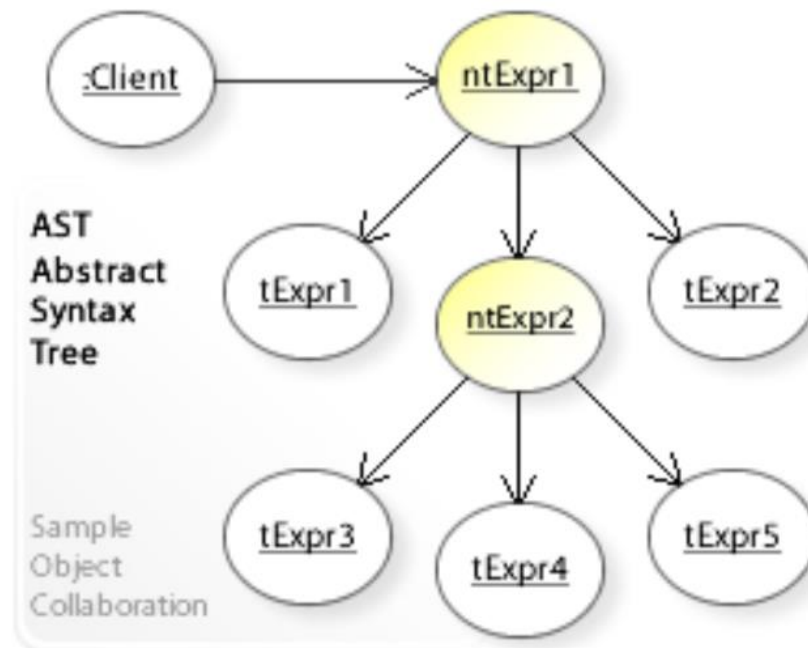
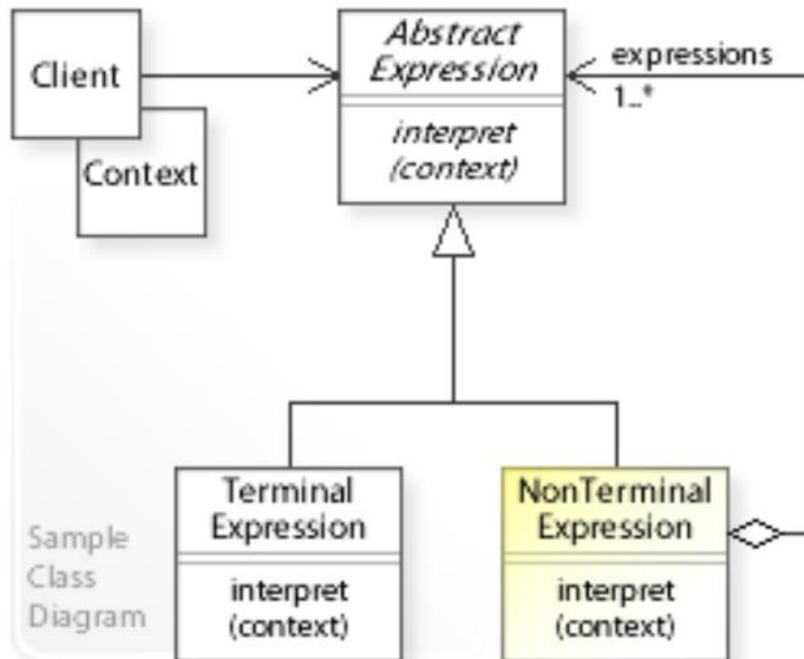
• •

• •

Using « Expression »
instead of « Double »
... advanced feature

Interpreter

Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.



REMINDER

Example of Interpreter : Math Expression

expression ::= plus | minus | variable | number
plus ::= expression expression '+'
minus ::= expression expression '-'
variable ::= 'a' | 'b' | 'c' | ... | 'z'
digit ::= '0' | '1' | ... | '9'
number ::= digit | digit number

```
abstract class Expression {}
```

```
class NumberExpression extends Expression {  
    double value;  
}
```

```
class VariableExpression extends Expression {  
    String name;  
}
```

```
class BinaryOperationExpression extends Expression {  
    Expression leftOperand;  
    String operator;  
    Expression rightOperand;  
}
```

Example, using JEP library

```
<dependency>  
  <groupId>org.scijava</groupId>  
  <artifactId>jep</artifactId>  
  <version>2.4.2</version>  
</dependency>
```

```
JEP jep = new JEP();
```

```
jep.addVariable("x", 123);
```

```
Node expr = jep.parseExpression("x+2");
```

```
Double eval = (Double) jep.evaluate(expr);
```

```
System.out.println("x=123,  x+2 => " + eval);
```

Reminder ... « Bridge » design-pattern

ANNEXE

```
public class DrawingExprContext {  
    protected Map<String, Object> values;  
    protected org.nfunk.jep.JEP internalImpl;  
    ..  
  
    public class DrawingExpr {  
        protected String text;  
        protected org.nfunk.jep.Node internalImpl;  
        ..  
    }  
}
```

