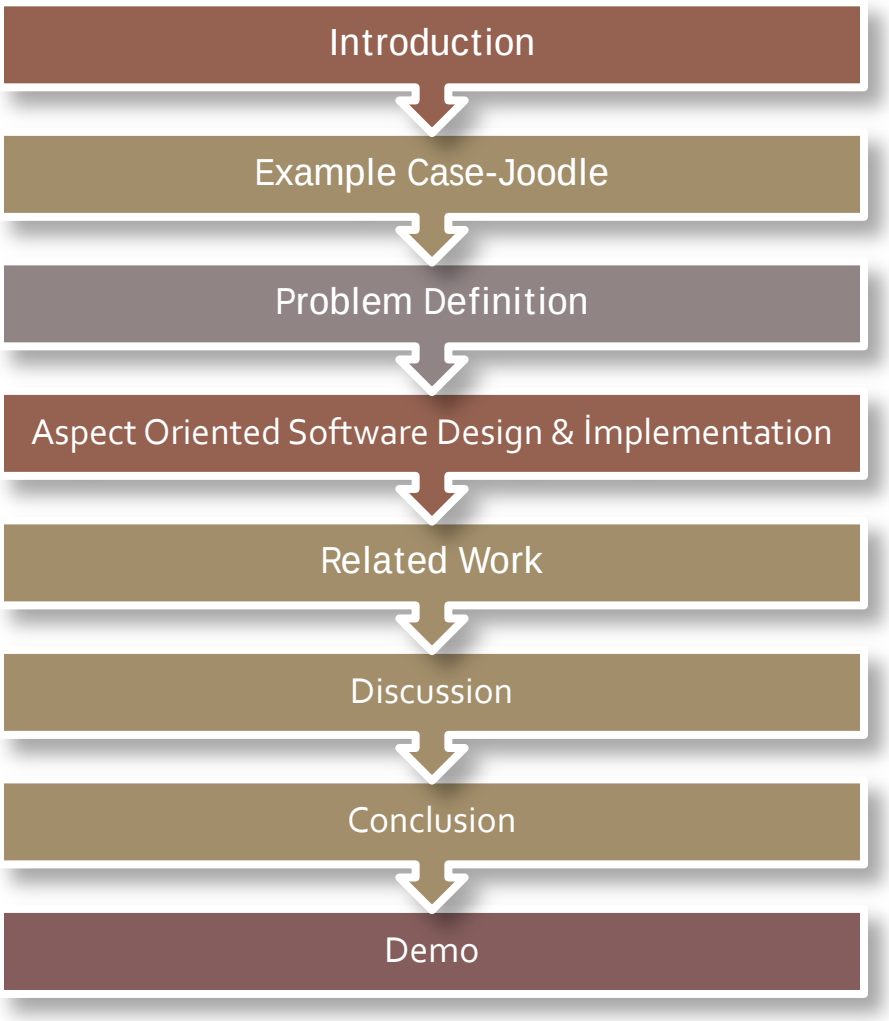
ASPECT-ORIENTED EXTENSION OF VECTOR IMAGE DRAWING TOOL JOODLE

Mehmet Ali ABBASOĞLU
Kemal EROĞLU
Aytuğ Murat AYDIN

Outline


Introduction
Example Case-Joodle
Problem Definition
Aspect Oriented Software Design & İmplementation
Related Work
Discussion
Conclusion
Demo

INTRODUCTION

Vector Graphics

Features

- Saves the data of shapes in mathematical formulations instead of pixels.
- Keeps the quality of the image
- Scaling does not distort the image

Used Areas

- Text
- Multimedia
- 3D rendering
- Many other information and art areas.

Background Information

- Instead of pixels, descriptions of the files are saved in XML.
- Vector Graphics are impressive alternatives to other image types
- SVG format is advocated by World Wide Web Consortium (W3C) for its benefits.

Vector Image Drawing Tools and AOSD

A good candidate for AOSD

- ❑Crosscutting Concerns
- ❑Scattering Classes and Tangling Code
- ❑High Dependency on Features
- ❑Inefficient design

Scattering Classes

Tangling

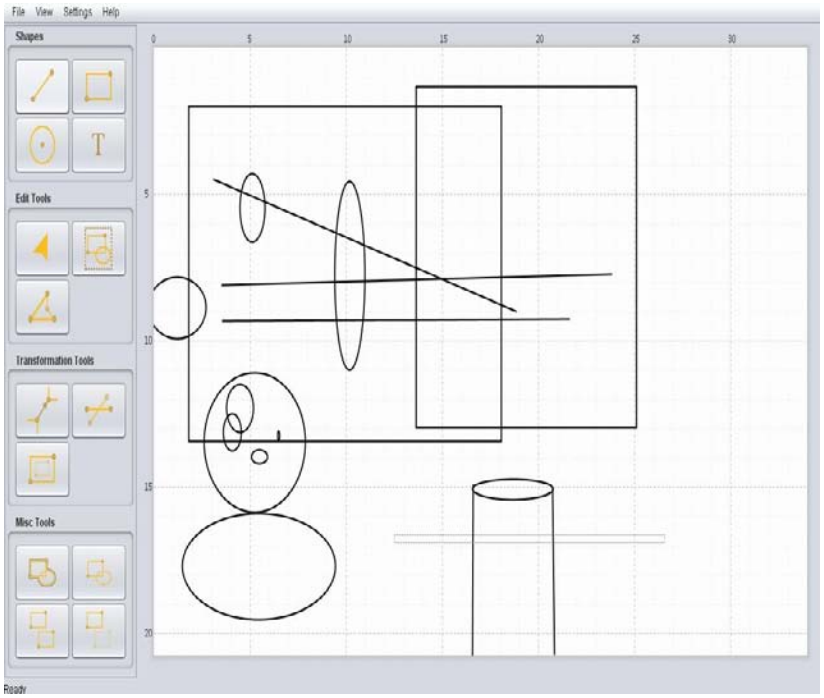
Dependency

EXAMPLE CASE - JOODLE

Example Case- Joodle

Overview of Joodle

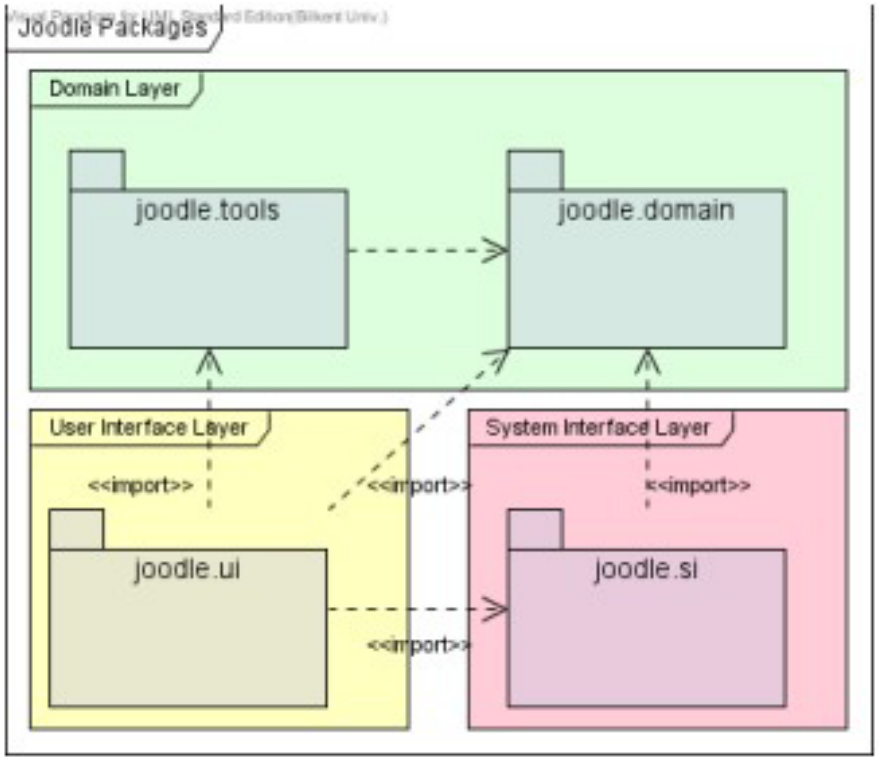
- A Vector Image Tool
- Uses SVG format
- Saves data into XML file
- Has pre-defined primitive shapes
- Implemented in Java



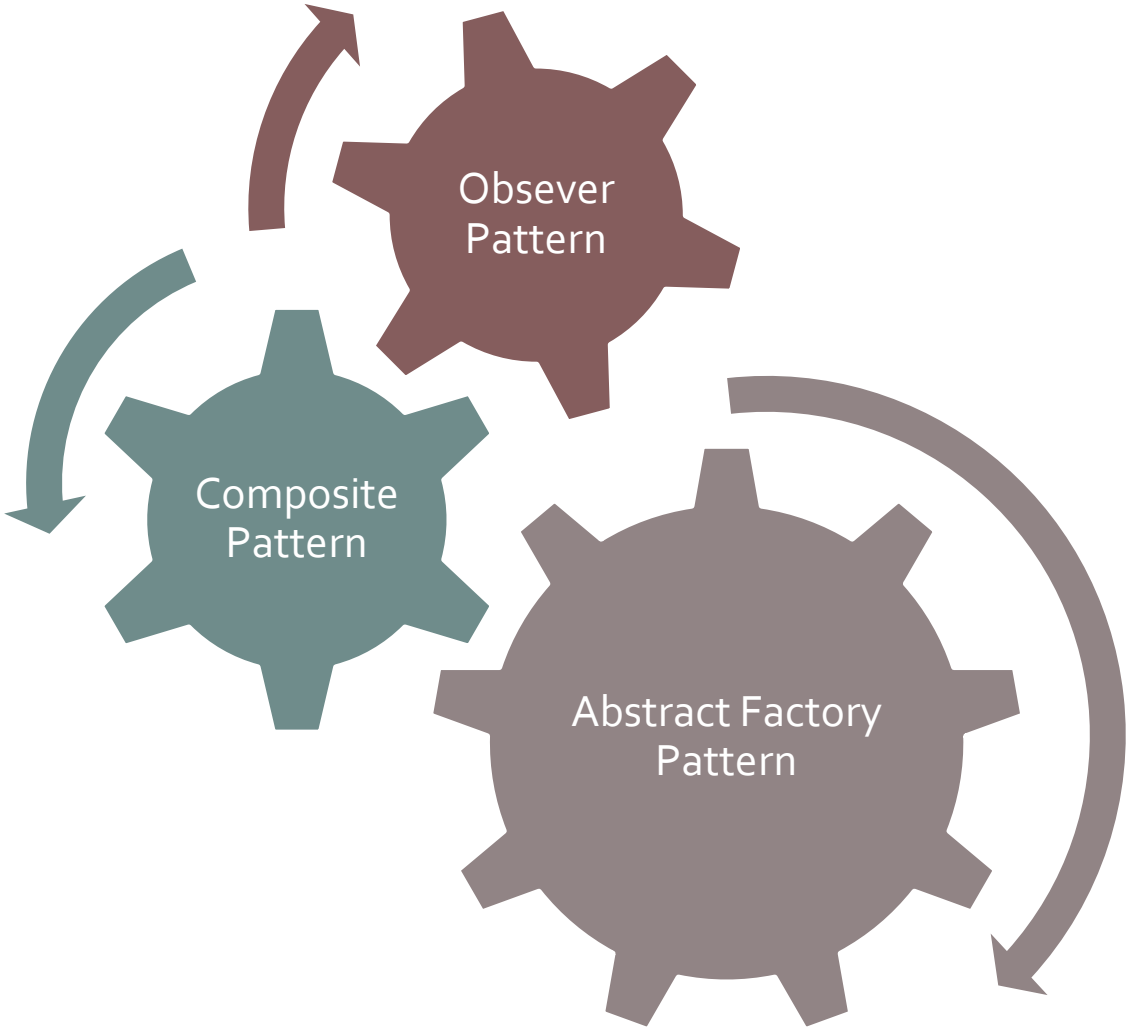
OO Design- Joodle

Existing Packages

- Tools Package
- Domain Package
- UI Package
- SI Package



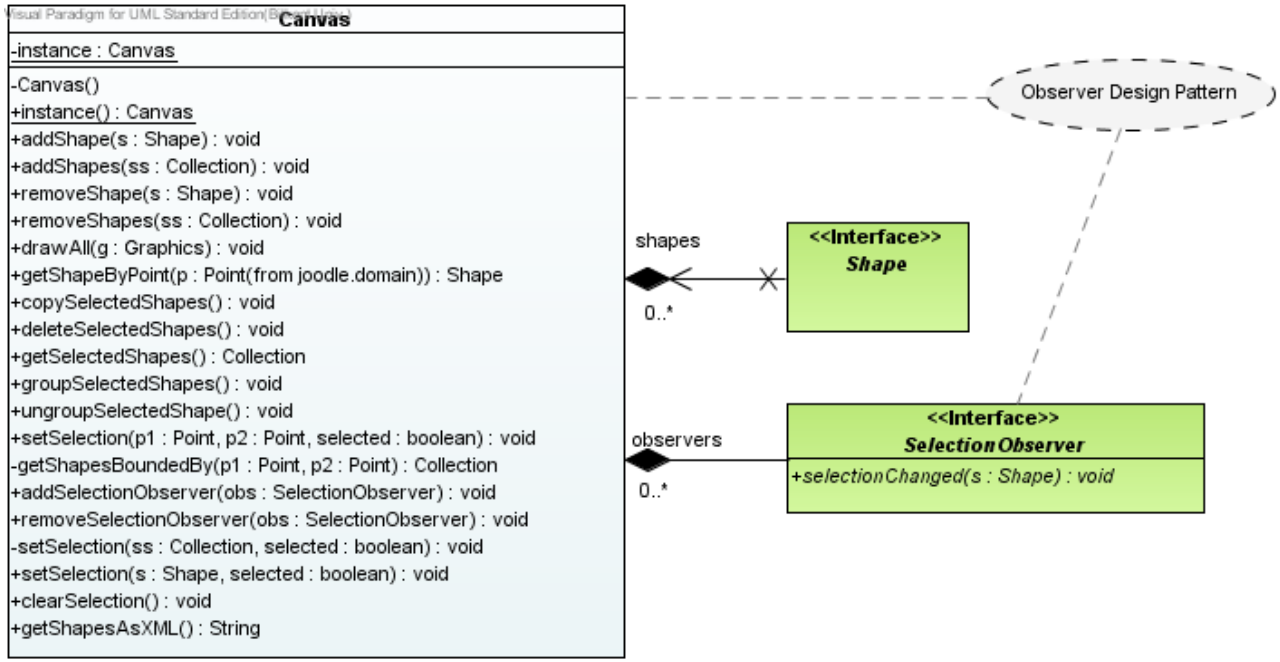
Applied Patterns- Joodle



Observer Pattern

Aim and Motivation

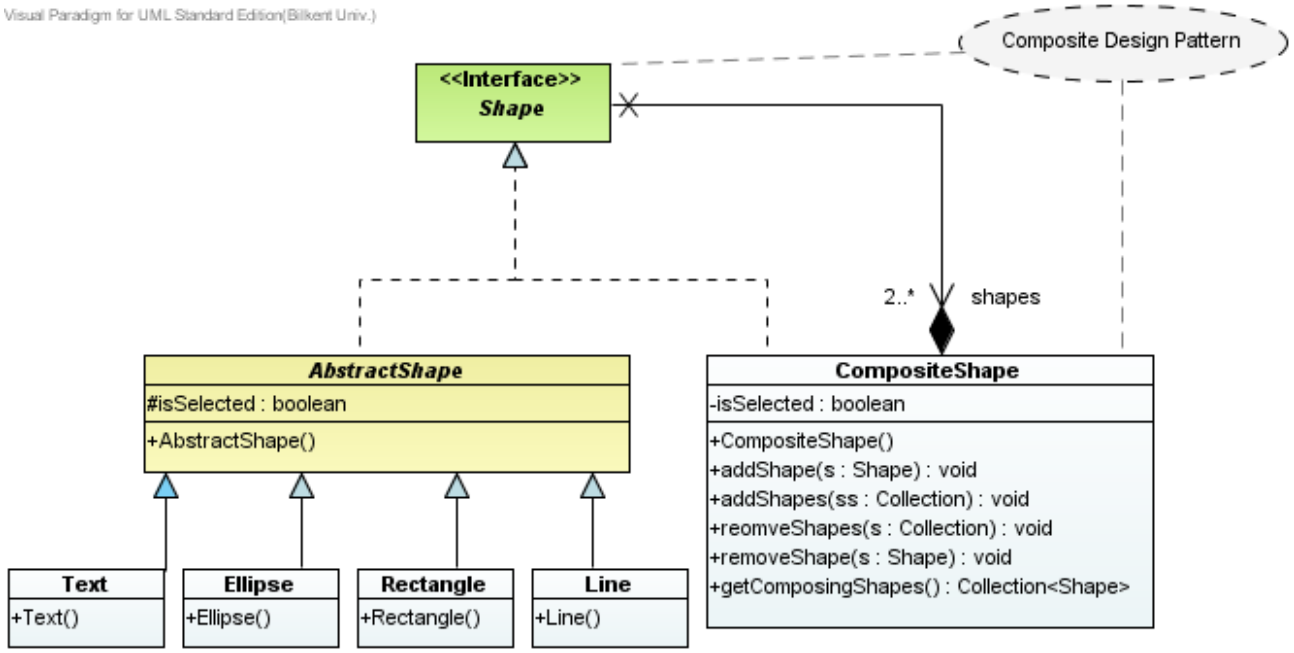
- Canvas, SelectionObserver are the classes of this pattern.
- Maintaining consistency between Canvas and other Shape classes



Composite Pattern

Aim and Motivation

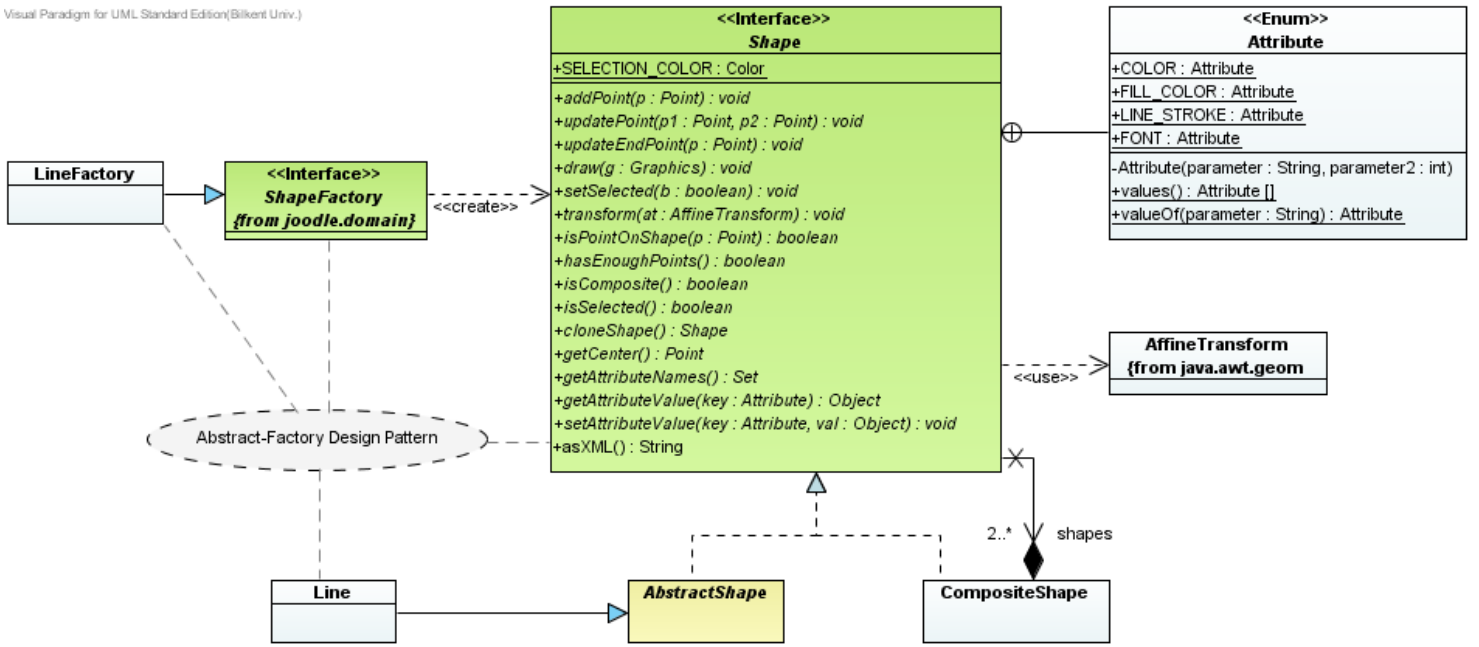
- Enabling the users to create complex shapes by using existing shapes
- Each shape can be used for creating larger shapes



Abstract Factory Pattern

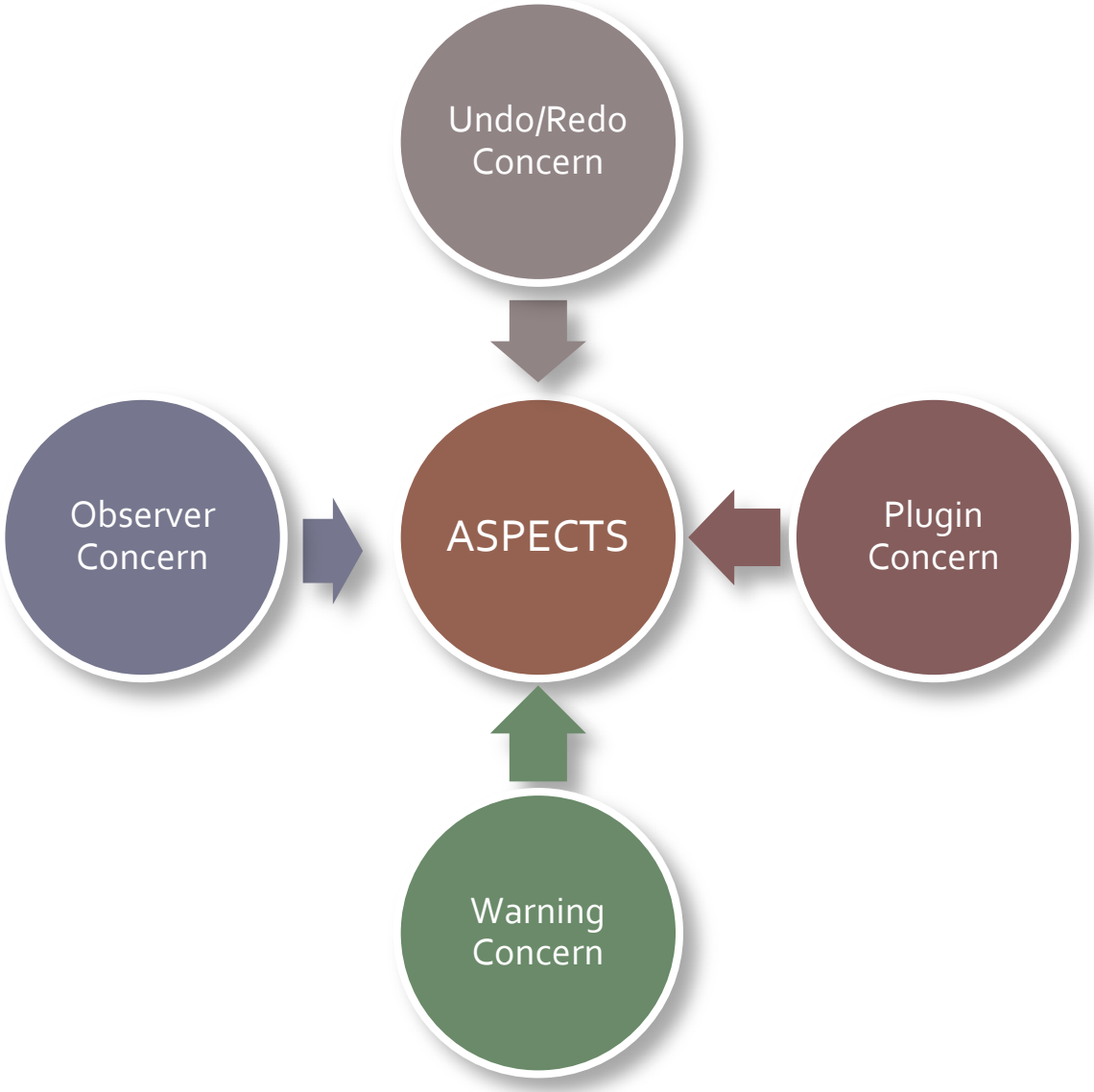
Aim and Motivation

- Creating a composite class which has all the members which have been created differently



CROSSCUTTING CONCERNS

Crosscutting Concerns



Observer Concern

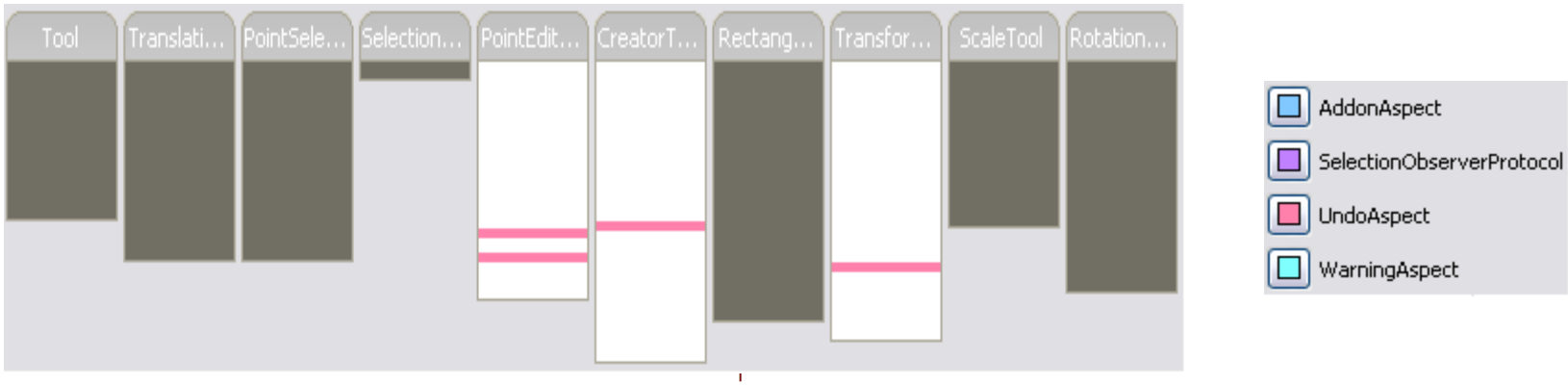
Main Problems

- Modularization Problem
- Changing classes deficiencies the code and pattern
- High tangling and low cohesion in classes of this pattern
- Dependency between classes

Undo / Redo Concern

Main Problems

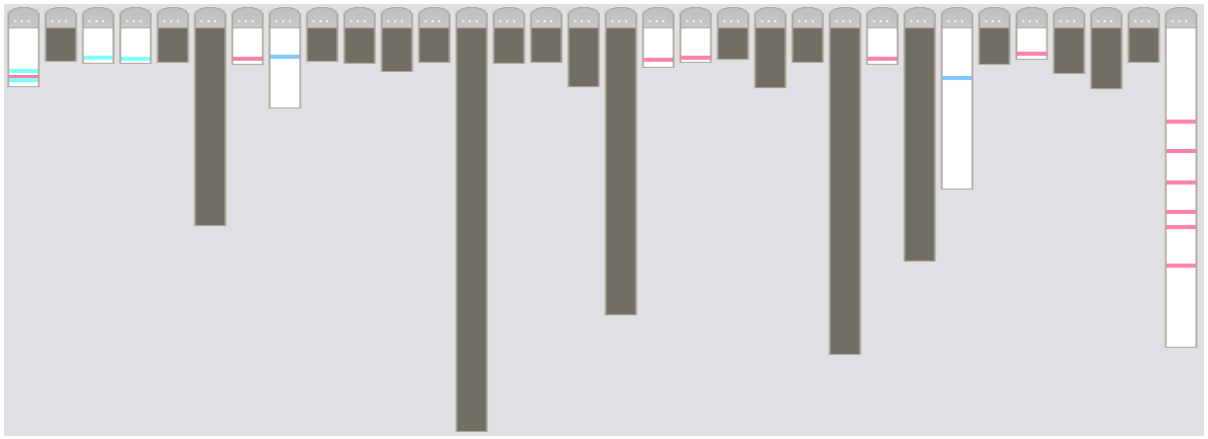
- Not Held in Joodle
- Creates inefficient use of the program
- Causes Loss of Data
- Needs modularization

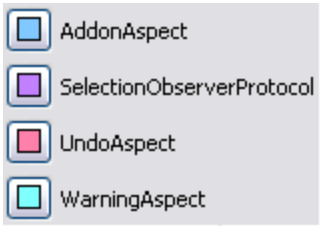


Add-on Concern

Main Problems

- Only pre-defined shapes handled
- More complex shapes may be needed
- Requires changing new classes or writing from scratch
- Ultimate cause is scattering and tangling code

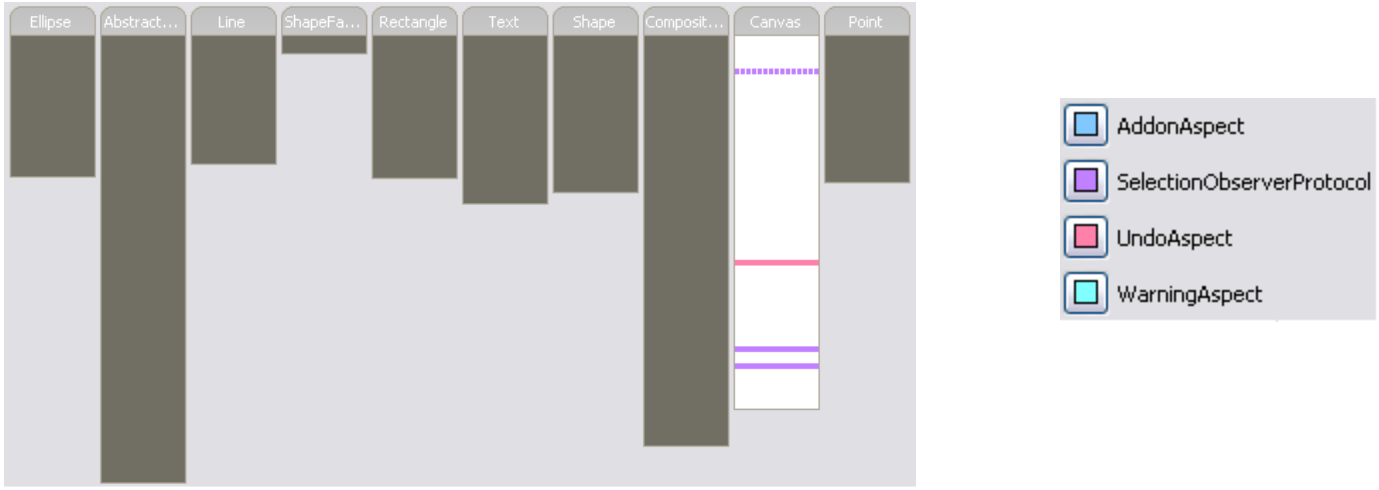




Warning Concern

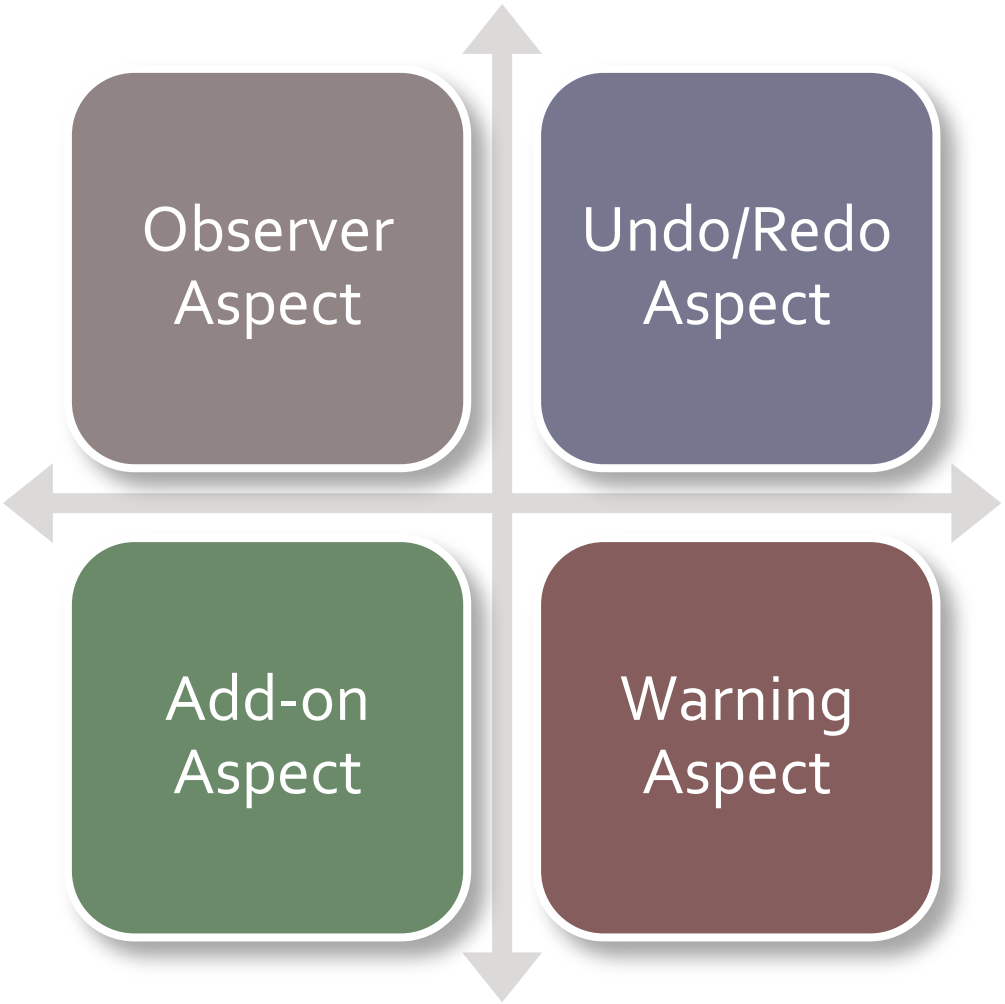
Main Problems

- Deficiencies in Code
- Error handling causes scattering classes
- Error handling classes are dependent to each other
- Each warning concern crosscuts multiple classes



ASPECT ORIENTED PROGRAMMING

Aspects


Observer Aspect
Undo/Redo Aspect
Add-on Aspect
Warning Aspect

AspectJ

Main Problems

- A general purpose AO programming language
- An Integrated extension to Java
  - Accepts all java program as input

Reason for Choice

- Popular
- Simple
- Integrated with tools(Eclipse, Netbeans)

Observer Aspect

Modularization

- No need to make amendments to protect Observer pattern during changes on classes
- Can be applied on other classes without creating dependency.
- Handled outside of the classes on a different aspect

```
1  package Aspect;
2  import java.util.ArrayList;
8
9  abstract aspect SelectionObserverProtocol
10 {
11     abstract pointcut selectionChanged( Shape s);
12
13     after(Shape s) : selectionChanged(s) {
14         Canvas.instance().fireSelectionChanged(s);
15     }
16
17     public Collection<SelectionObserver> Canvas.observers = new ArrayList<SelectionObserver>
18
19     /**
20      * Adds an observer to the set of observers
21      * @param so the observer to be added
22      */
23     public void Canvas.addSelectionObserver(SelectionObserver so) {
24         if(!observers.contains(so))
25             observers.add(so);
26     }
27
28     /**
```

Undo/Redo Aspect

This function is handled without making alterations on existing code

- No dependent classes
- Low tangling high cohesion between classes
- This function can be applied to other classes without making amendments on them

```
18 public aspect UndoAspect
19 {
20
21     public ArrayList<String> undoList = new ArrayList<String>();
22     int index = 0;
23     boolean undo= false;
24
25
26     // SAVE PROCESS AFTER AN OPERATION
27     pointcut canvasChanged():
28         (withincode(* TransformationTool.handleRelease(..)) && call(* Shape.transform(..)))
29         (call(* Canvas.copySelectedShapes())) ||
30         (call(* Canvas.deleteSelectedShapes())) ||
31         (withincode(* Canvas.groupSelectedShapes(..)) && call(* Canvas.addShape(..))) ||
32         (call(* Canvas.ungroupSelectedShape())) ||
33         (withincode(* CreatorTool.handlePress(..)) && call(* Canvas.addShape(..))) ||
34         (withincode(* PointEditTool.handleRelease(..)) && call(* foo(..))) ||
35         (withincode(* PointEditTool.handleKeyPress(..)) && call(* foo(..)));
```

Add-on Aspect

- It is a necessary function for the application
- Needs to be created without creating deficiencies in the implemetation
- New shape types added to the system without changing and creating new shape types
- No need for creating new classes for new shape types

```
package Aspect;
import java.io.File;

public aspect AddonAspect
{
    pointcut addonFound():

    after() returning(ArrayList<ShapeFactory> sf):addonFound()

    * By defaults add-ons are in the current directory
    public static String addonsDir = ".";

     * Sets the add-ons' directory to the given new directory
    public void setAddonsDir(String dirName) {

     * Tries to instantiate a Shape from the class
    private static Shape makeInstance(URL u, String cn)

    public static ArrayList<ShapeFactory> findAddons
}
```

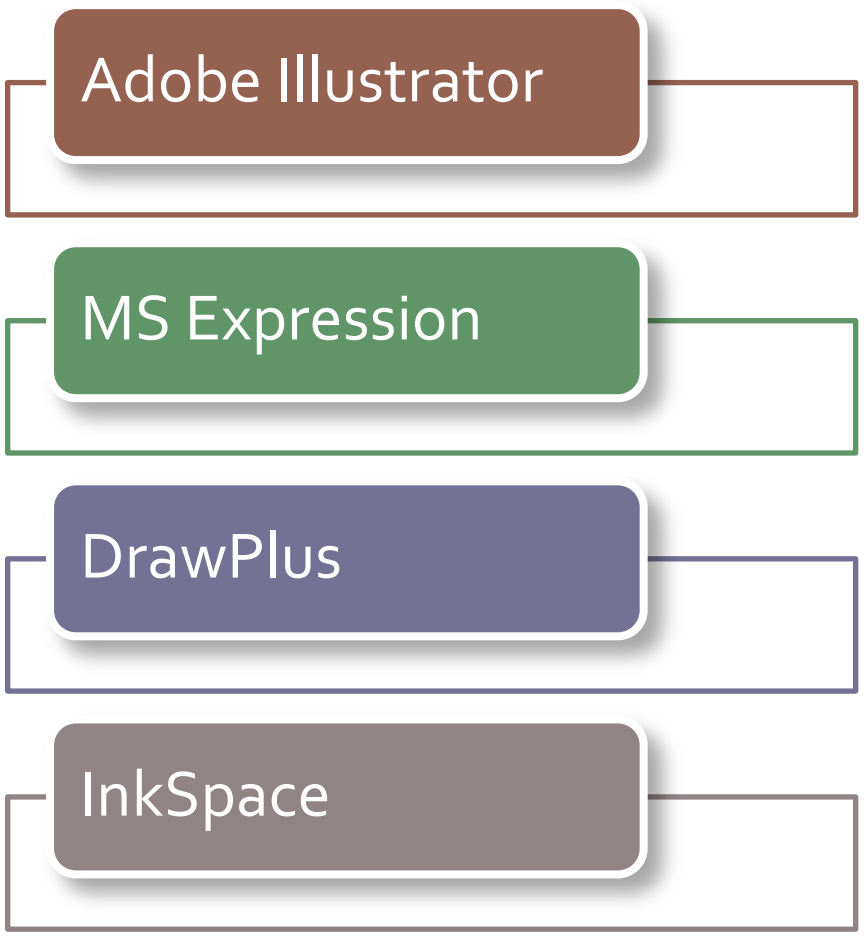
Warning Aspect

- Possible errors are handled on save issues.
- Load work and Save work issues are handled without causing tangling and scattering

```
public aspect WarningAspect
{
    // EXIT IMAGE WARNING
    pointcut exitImage():

    before(): exitImage()

    // NEW IMAGE WARNING
    pointcut newImage():

    before(): newImage()

    // LOAD IMAGE WARNING
    pointcut loadImage():

    before(): loadImage()

    // LOAD ERROR HANDLING
    pointcut loadError():

    before(): loadError()

    // SAVE ERROR HANDLING
    pointcut saveError():

    before(): saveError()
}
```

RELATED WORKS

Related Work



- But none of them implemented with AOSD

CONCLUSION

Conclusion

Modularization

- Good way to handle crosscutting concerns
- Creating high cohesion and low tangling
- Modularization
- Lowering deficiencies in implementation
- Efficient for the implementation

Vector Graphics Tool

- Undo/Redo Aspects beneficial.
- Add-on aspect is promising in terms of development.
- New improvements are inevitable but AOPD makes it easier.

References

- Scalable Vector Graphics, Retrieval Date: 14.12.2009
- http://www.w3.org/Graphics/SVG/
- E. Gamma, R. Helm, R. Johnson and J. Vlissides. Design Patterns Elements of Reusable Object Oriented Software. Addison-Wesley, 2008
- Jacob Borella, The Observer Pattern Using Aspect Oriented Programming, IT University of Copenhagen, 2003
- Ivan Kiselev, Aspect Oriented Programming with AspectJ, Sams Publishing, 2003
- [6] The AspectJ(TM) Programming Guide. Retrieval Date: 14.12.2009 http://www.eclipse.org/aspectj/doc/released/progguide/

DEMO

Thank you for listening.

Any questions and comments are appreciated