An on-line library of design patterns

Weizhong Wu

Follow this and additional works at: https://scholarworks.lib.csusb.edu/etd-project

Part of the Databases and Information Systems Commons

Recommended Citation

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.
AN ON-LINE LIBRARY OF DESIGN PATTERNS

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Wei-Chung Wu
June 2004
AN ON-LINE LIBRARY OF DESIGN PATTERNS

A Project
Presented to the
Faculty of
California State University,
San Bernardino

by
Wei-Chung Wu
June 2004

Approved by:

Dr. Arturo Concepcion,
Chair, Computer Science

Dr. Ernesto Gomez

Dr. Kay Zemouchi

04-Jun-2004 Date
ABSTRACT

Katrina Y. Ji [4] proposed in her Master’s thesis the ADAP (Applying Designs and Patterns) Model to provide guidance for software engineers who utilize design patterns for developing E-Commerce applications. She proposed a transition between the abstract level and implementation level with the following levels of abstraction: concrete design patterns (by using DisCo)[5], specific design patterns (UML), and integrated design patterns. The ADAP model provides a basis for automating design patterns applications and developing tools for applying design patterns. This project, Online DPL (Design Patterns Library), is one of a series of tools to be developed in the ADAP model.

DPL provides storage and retrieval of design patterns for the purpose of using them in software development. DPL provides a set of six Gamma[2] generic design patterns and one domain-specific design pattern as a base set of design patterns from which a software engineer can immediately use. DPL has drawing capabilities that allows the creation and modifications of design patterns in the DPL. In summary, the on-line DPL aims to assist a software engineer in using the library of design patterns. In addition, DPL is accessible on the Internet.
ACKNOWLEDGMENTS

The support of the National Science Foundation under award 9810708 is gratefully acknowledged.
TABLE OF CONTENTS

ABSTRACT .................................................................................. iii
ACKNOWLEDGMENTS ......................................................... iv
LIST OF FIGURES ................................................................. vii

CHAPTER ONE: SOFTWARE REQUIREMENTS SPECIFICATION

1.1 Introduction ................................................................. 1
1.2 Purpose of the Project .............................................. 3
1.3 Scope ........................................................................... 4
1.4 Definitions and Acronyms ......................................... 5
1.5 Design Pattern Library Overview ............................. 8
1.6 Specific Requirements ................................................. 15
1.7 Organization of the Thesis ........................................... 28

CHAPTER TWO: SOFTWARE DESIGN

2.1 Design Pattern Library Design .................................. 29
2.2 Detailed Design ............................................................ 31

CHAPTER THREE: SOFTWARE QUALITY ASSURANCE

3.1 Introduction ................................................................. 42
3.2 Test Plans ...................................................................... 42

CHAPTER FOUR: MAINTENANCE

4.1 Introduction ................................................................. 62
4.2 File Structure .............................................................. 62
4.3 Recompile ...................................................................... 63
4.4 Installation Process ...................................................... 64
CHAPTER FIVE: CONCLUSIONS AND FUTURE DIRECTIONS

5.1 Summary of Results ........................................... 65
5.2 Future Directions ........................................... 66

APPENDIX A: JAVA SOURCE CODE ............................... 68
APPENDIX B: HTML CODE ......................................... 126
REFERENCES ....................................................... 138
LIST OF FIGURES

Figure 1. Applying Designs and Patterns Model ............. 2
Figure 2. Structure of Observer Pattern .................. 8
Figure 3. Sequence Diagram of Observer Pattern .......... 9
Figure 4. Distributed Cooperation Specification of Observer Pattern .................................. 10
Figure 5. Structure of Mediator Pattern ................... 11
Figure 6. Distributed Cooperation Specification of Mediator Pattern .................................. 12
Figure 7. Product Perspective ................................. 13
Figure 8. Welcome Page of Design Pattern Library ......... 16
Figure 9. External Interface-display Diagram of Design Pattern Library ........................................ 19
Figure 10. Open Dialog to Open a Text File ................ 20
Figure 11. Save Text File Dialog ............................... 21
Figure 12. Class Diagram of Master-slave Pattern ........ 22
Figure 13. Create/Modify User-created Diagram of Design Pattern ........................................ 23
Figure 14. Store User-created Diagram ........................ 24
Figure 15. Objects Interaction (Structure) of Mediator Pattern in DPL ....................................... 25
Figure 16. Distributed Cooperation Specification of Mediator Pattern in Design Pattern Library ...... 26
Figure 17. Class Diagram of Design Pattern Library ....... 30
Figure 18. Class Diagram of GUI0 Class .................... 32
Figure 19. Class Diagram of DescriptionPanel10 Class ...... 35
Figure 20. Class Diagram of DrawingCanvas0 Class ....... 36
Figure 21. Class Diagram of DrawableInterface ............. 38
Figure 22. Class Diagram of ToolBox0 Class ..................39
Figure 23. Class Diagram of ToolButton .......................40
Figure 24. Class Diagram of WindowCloser0 Class ..........41
Figure 25. External Interface .................................43
Figure 26. Open a Text File .................................44
Figure 27. Opened Text File (Distributed Cooperation Specification) ..................45
Figure 28. Save File Dialog .................................46
Figure 29. Submenu Showing Observer Sequence Diagram ......47
Figure 30. Opened Observer Pattern Sequence Diagram ......48
Figure 31. An Observer Design Pattern Class Diagram is Opened ........................................49
Figure 32. The Snapshot of Drawing Functionality ..........50
Figure 33. User Inputs Text “Mediator” and “ConcreteMediator” .................................51
Figure 34. User Types “Colleague” .............................52
Figure 35. User Types “ConcreteColleague1” and “ConcreteColleague2” .......................53
Figure 36. User Click on the “Class” Button .................54
Figure 37. Drawing the Inheritance Relationship Between Classes ........................................55
Figure 38. Reload Image File Dialog ..........................56
Figure 39. A Reloaded Mediator Pattern Class ..............57
Figure 40. User Types in Attribute and Click on Attribute Button to Store the Attribute and Do the Same for the Method ........................................58
Figure 41. A Class Diagram of Mediator Pattern has been Modified ..................................59
Figure 42. The User Inputs More Attributes and Methods and Then Clicks on the “Modify” Button ......60
Figure 43. Save File Dialog is Opened After the User Click the "Save Image" Button .................61
CHAPTER ONE
SOFTWARE REQUIREMENTS SPECIFICATION

1.1 Introduction

Katrina Y. Ji [4] developed the ADAP (Applying Designs and Patterns) Model, in her Master’s thesis. There are five levels in the ADAP model, see Figure 1.1. The first level is generic/domain-specific design patterns level, which is the abstract level of design patterns. The second level is concrete design patterns level which specifies the design patterns to the level of implementation where they will be used. Design patterns can be concretized with DisCo[5] language. The third level is specific design patterns level. After design patterns are concretized, they can be modified and renamed to be specific design patterns. The fourth level is integrated design patterns level. Design patterns could be integrated or combined to create a new design pattern. The fifth level is the implementation level. Integrated design patterns are implemented in specific programming languages such as Java, C++, or Smalltalk language. DPL provides a tool that allows users to choose a generic or domain-specific design pattern to be used in software development by proceeding in steps from the abstract level to the implementation level. DisCo specification is used in concrete design patterns in the ADAP model will be provided in DPL. After concretizing design patterns, ambiguities are
eliminated in the generic design patterns. As a result, we need to consider specific requirements for specific applications thus the concrete design patterns can be modified to fit into the environment of the specific system. Specific design pattern is represented in UML (Unified Model Language) class diagrams. This will be provided in DPL. The specific design patterns are detailed software design before the code.
implementation. The integrated patterns level in the ADAP Model shows the combined patterns in an application system. After the specific design patterns level, the design patterns are ready to be implemented.

1.2 Purpose of the Project

The Design patterns are a promising technique to solve recurring problems that arise when building software in domains like business data processing, telecommunication, graphical user interfaces, databases, and distributed communication software. Design patterns capture the static and dynamic structures and collaborations of components in successful solutions and provide software engineers a way to think about designs on a high level of abstraction instead of thinking of individual classes and their behaviors.

Using design patterns is considered an effective way to develop reusable software. There are 23 design patterns presented by Gamma [2]. They are typically given in terms of programming-level abstractions or by using presentations that lack rigorous definitions of temporal behaviors. We can not reason the temporal properties of patterns unless we go to details of implementation level. If we wish to reason about the temporal behaviors of the system, even when the specification is incomplete or abstract, we need to formalize design patterns. For pattern-oriented software development, formalization is used to express temporal behaviors of design
patterns and the systems using design patterns as their skeletons. In order to formalize the temporal behaviors of design patterns, DisCo(Distributed Cooperation) [5] method was used. The DisCo method allows rigorous reasoning and also provides an analytic view to the system being designed. Thus, when we want to compose DisCo specifications of complex system, we just combine patterns that have already been formalized.

1.3 Scope

The on-line library of design patterns in DPL includes both a subset of generic and one domain-specific design patterns such as master-slave pattern. This project is proposed to develop an on-line library to help software engineers to translate generic design patterns to concrete design patterns. The graphical user interface was developed using Java and allows the user to develop their own design patterns by modifying/changing stored design patterns to create new design patterns. The user can store their own design patterns in a GUI application.

What Design Pattern Library can do:

- Provides UML diagrams of six generic design patterns including factory design patterns, builder design pattern, composite design pattern, flyweight design
pattern, observer design pattern, and mediator design pattern.

• Provides documentations for the above design patterns in UML class diagrams, sequence diagrams, and DisCo specification.

• Provides UML diagram of one domain-specific design patterns - master-slave pattern.

• Provides drawing tools to create new UML diagrams or modify existing/stored UML diagrams.

What Design Pattern Library cannot do:

• Reverse engineering (Detect underlying design patterns) for a E-Commerce application (not yet developed).

• Combine or integrate design patterns (not yet developed).

1.4 Definitions and Acronyms.

The definitions, acronyms, and abbreviations used in the document are described in this section.

Design Patterns

Design patterns[2] are a promising technique for solving recurring problems that arise when building software in domain like business data processing, telecommunication, graphical user interfaces, databases, and distributed communication software. Design pattern capture the static and dynamic
structures and collaborations of components in successful solutions and let software engineers a way to think about designs on a high level of abstraction instead of thinking of individual classes and their behaviors.

**Distributed Cooperation**

Distributed Cooperation (DisCo) is an object-oriented method for specifying and reasoning about the reactive and distributed systems. The method incorporates a specification language, a methodology for developing specifications using the language, and a graphical tool support for the methodology.

**Domain-Specific Design Patterns**

Domain-Specific Design Patterns are core solution to problems in a specific domain. They are similar to generic Design Patterns and are at the same level of abstract level as Generic design patterns. Domain-Specific Design Patterns are only applicable in a certain domain application area.

**Generic Design pattern**

Generic design patterns are descriptions of solutions in natural languages. They are generally published design patterns, and they are applicable in all area of application.

**The Apply Design And Pattern Model**

The Apply Design And Pattern (ADAP) Model was proposed by Katrina Y. Ji [4] to provide guidance for software engineers to utilize design patterns. She proposed a
transition between abstract level an implementation level with concrete design patterns (by using DisCo), specific design patterns (UML), and integrated design patterns. The ADAP model provides a basis of automating design patterns applications and developing tools for applying design patterns. "Without automating the whole processes, applying design patterns will be laborious and error-prone, particularly for complex system" [4].

**Temporal Logical OF Actions (TLA)**

TLA is a logic for specifying and reasoning about concurrent and reactive systems. TLA may be used for reasoning about DisCo specifications. The incremental specification methodology using superposition guarantees preservation of safety properties. A concurrent algorithm is usually specified with a program. Correctness of the algorithm means that program satisfies a desired property. We proposed an approach that both algorithm and property are specified by formulas in a single logic. Reasoning about 5000 lines of C code would be a very difficult and time-consuming task, but we can reason about one-page abstract algorithm. If the algorithms we reason are not real, compilable programs, then they do not have to be written in a programming language. Rigorous reasoning is the only way to avoid subtle error in concurrent algorithm.
1.5 Design Pattern Library Overview

DPL will allow a user to click on a submenu to choose a pattern, and display the structure of a specific pattern, sequential diagrams, or the DisCo specification (Actions & Relations). The detailed documentation of each pattern will be displayed. DPL allows a user to create generic design patterns by modifying existing ones and storing it back to DPL.

Product Prospective

We describe two example design patterns in this section to illustrate how design patterns in DPL will be displayed. The first example is the observer design pattern.

Observer is a pattern, which defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. See Figure 2 for the class diagram of the

![Figure 2. Structure of Observer Pattern](image-url)
observer pattern. Figure 3 shows the sequence diagram of the observer pattern.

![Sequence Diagram of Observer Pattern](image)

Figure 3. Sequence Diagram of Observer Pattern

Figure 4 shows the DisCo specification of the observer pattern. Action Notify denotes that the content of a subject have been modified. Action Update represents a transmission of modified data from a subject to an observer.
DisCo Specification is the following:

Class Subject = { Data }  Class Observer = { Data }
Relation (0..1). Attached . (*) : Subject X Observer
Relation (0..1). Updated . (*) : Subject X Observer

Action

Attached (s: Subject ; o: Observer):  
~s. Attached.o
->s. Attached’.o

Detached (s: Subject ; o: Observer):
  s. Attached’.o
-> ~ s. Attached’.o
^-~s. Updated’.o

Notify( s: Subject, d):
  -> ~s. Updated’. class Observer ^ s.Data’ =d

Update (s: Subject ; o*: Observer):
  s. Attached.o ^ ~s.Updated.o ^ d= s.Data
  -> s. Updated’.o ^ o.Data’=d

Figure 4. Distributed Cooperation Specification of Observer Pattern

The second example is the mediator design pattern.

Mediator pattern defines an object that encapsulates how a set of objects interact. See Figure 5 for the UML class diagram and Figure 6 for the DisCo specification of Mediator pattern. There is a set of colleagues that connects a mediator.
Connected colleagues can communicate with each other via putting and receiving message through the mediator.

Figure 5. Structure of Mediator Pattern
Mediator pattern can be expressed in DisCo language as

Class Colleague.

Class Mbox = { data }.

Relation (0..1) . Sender (0..1) : Mbox Colleague.
Relation (*) . Receiver (*) : Mbox Colleague .

Class Mediator = {{ Boxes} : Mbox }.

Relation (0..1) . Connected (*) : Mediator Colleague.

Connect(m: mediator, mb: Mbox ; c: Colleague):

~ m.Connected .c ^ ~ mb.Sender. class Colleague
^ mb m.Boxes

->m.Connected ' .c ^ mb.Sender'.c

Put (m:Mediator ; mb: Mbox ; from , {to} ; Colleague ; d): m.Connected.from ^ m.Connected.to
^ m.Sender. from ^mb m.Boxes

-> mb.Receiver'. to ^ ~ mb.Receiver' . (class colleague - to) ^ mb. data' d.

d denotes a set of suitable receivers.

(class colleague - to ) refers to those instances that do not belong to set {to} .

Get(m:Mediator ; mb: Mbox ; c*: Colleague ; d):

m.Connected. c ^ mb.Receiver.c
^mb belong to m.Boxes
^ mb.Message = d

⇒ ~ mb. Receiver'.c

Figure 6. Distributed Cooperation Specification of Mediator Pattern

Product Function

Figure 7(a) shows the functions of DPL represented using Use Case Diagram and shown in Figure 7(b) is the deployment diagram for the Online DPL.
(a) Use Case Diagram of Design Pattern Library.

(b) System interface of Online DPL.

Figure 7. Product Perspective

Summary of Product Functions
1. Allow the user to view and display all the design patterns stored in DPL, including all its documentation.
2. Allow the user to utilize concrete design patterns and
save time in building necessary concrete design patterns for
the next step of the specific design patterns in the ADAP
model.

3. Allow the user to create new concrete design patterns
by modifying existing concrete design patterns using
refinements in the DisCo specification. The user can store
his/her own created concrete design patterns in their local
files.

DPL on-line library can only be browsed but a software
engineer can copy any design patterns in DPL and store and
modify them in the user’s local directories.

User Characteristics

The on-line library is intended for software engineers,
such as C++ programmers, to take advantage of concretized
design patterns and proceed to specific design patterns. These
specific design patterns will be represented using UML. The
user can then integrate the design patterns and finally
implement the design patterns in a programming language. The
DPL user is supposed to be familiar with object-oriented
paradigm and knowledgeable in object-oriented programming
language such as C++, Java or Smalltalk. This DPL is intended
to help software developers to utilize design patterns.

Constraints

Due to the complexity of combining DisCo specification
such as (formalization of specific design pattern, or
combination of two or more design patterns and also building an on-line library, DPL will not be able to implement all the design patterns mentioned in Gamma [2], only six of the 23 design patterns is stored. One domain-specific design patterns such as master-slave pattern, which could be used in E-commerce application, is stored in DPL.

Assumption and Dependency

DPL assumes the user has a Microsoft Internet Explorer 5.0 or later version or Netscape navigator 4.0 or later version browser supporting JavaScript.

1.6 Specific Requirements

External Interface

The URL for DPL is http://dpl.ias.csusb.edu/index.html
see Figure 8 for welcome page of DPL.
In DPL, there are several types of window user interfaces. UML class diagrams, sequence diagrams, and DisCo language will be used to specify and document specific patterns. The subMenu will be provided for a user to store and retrieve their own design patterns. The basic DPL GUI contains a title bar at the top of the window. There are menus that allow the user to select generic design patterns and domain-specific design patterns. When the user clicks on either the generic design pattern or the domain-specific design pattern subMenu, the contents of these design patterns will be
displayed. For instances, there are 6 design patterns[2] in the generic design patterns category and all of them can be shown. The detailed documentation of each pattern will be provided in the DPL library. The user can double click DPL1.exe in the attached CD to open this stand alone application Figure 8 shows the external interface of DPL. It also shows the product functions to display the UML diagrams of the design patterns.

The following is the description of the menu:

Under the "Creational" menu, the user can retrieve the use case diagram, class diagram, sequence diagram and DisCo specification of Factory Method design pattern and Builder design pattern.

Under the "Structural" menu, the user can retrieve user diagram, class diagram, sequence diagram and DisCo specification of Composite design pattern and Flyweight design pattern.

Under the "Behavioral" menu, the user can retrieve the use case diagram, class diagram, sequence diagram, and DisCo specification of Observer design pattern and Mediator design pattern

Under the "Domain-Specific" menu, the user can retrieve the use case diagram, class diagram, sequence diagram, and DisCo specification of Master-Slave design pattern.
"Combination" menu contains the sub menu for potential combination of different design patterns.

"Reverse Engineering" contains sub menu for detecting underlying design patterns.

"About" contains the help topic for DPL.

The following is a brief description for the buttons in the toolbar under menu bar: (See Figure 9)

"Open" button: To open a text file.

"Save" button: To save a text file.

"Clear" button: To clear the text in the content panel.

"Copy" button: To copy the text in the content panel.

"Cut" button: To cut the text in the content panel.

"Paste" button: To paste the text in the content panel.

"Store", "Retrieve" and "Home" buttons does not work in this browse only environment.

To save and retrieve existing image, the user can double click DPL2.exe in the attached CD to open the drawing application.

"Save Image" button to save the image.

"Reload Image" button to reload image and modify.

See Figure 13 for the "Reload Image" and "Save Image" button.
Figure 9. External Interface-display Diagram of Design Pattern Library

To open a text file, click on "Open" button in toolbar. An "Open" file dialog will displayed for the user to choose a text file. The snapshot of this GUI (graphic user interface) is shown Figure 10.

To save a text file, click on "Save" button in toolbar. A "Save" file dialog will displayed for the user to choose a text file. The snapshot of this GUI (graphic user interface) is shown Figure 1.11.
Figure 10. Open Dialog to Open a Text File

Figure 12 shows class diagram of a domain-specific design pattern (Master-Slave pattern).
The Combination of Observer and Mediator in DisCo specification.

```java
class Manager = Mediator.
class MgdSubject = Subject + Colle
class MgdObserver = Observer + C
MgdConnect(mgr:Manager, mb: Mb
refines Connect(m:Mediator = mgr;
for m belong to Manager, c belong t
Register(ms: MgdSubject; mo: Mgd
```

Figure 11. Save Text File Dialog

Figure 13 shows one of the product functions—to create/modify UML diagram of design patterns.

Figure 14 shows one of the major product function of DPL-Store user-created UML diagram of design pattern into DPL.
Figure 12. Class Diagram of Master-slave Pattern
Figure 13. Create/Modify User-created Diagram of Design Pattern
Figure 14. Store User-created Diagram

Figure 15 shows the objects interaction (Structure) of Mediator pattern in DPL.
Figure 15. Objects Interaction (Structure) of Mediator Pattern in DPL

Figure 16 shows the DisCo language of Mediator pattern in DPL.
DPL allows developers to access DPL on-line to browse different aspects of a pattern. In contrast, the application (GUI) will allow a user to draw his own generic pattern and store to his local files. The drawing stored by user can be reloaded later. DPL will run on any platform containing Web browsers that support HTML and JavaScript.

On the client side, a user would need Web browser such as Microsoft Internet Explorer version 5 (including later
version) or Netscape Navigator version 4 (including later version) to access the DPL.

Allow client to click on "Line" button to draw a line, click on "class" button to dynamically draw a class, click on "Inheritance" button for Inheritance relationship between classes. Click on "Text" button for text input, Click "Aggregation" for Aggregation relationship between classes. Modify the attribute and method of a specific class diagram, then store in the client user’s hard drive and reload the diagram later on.

There is also an on-line Website for DPL, but is available only for browsing the use case diagram, class diagram, sequence diagram and DisCo specification of a specific design pattern, they cannot be modified on line. DPL also provide the functionality for user to modify DisCo specification.

Performance Requirements

The Users is advised that he should download Java™ 2 Standard Edition Runtime Environment version 1.4. It has the latest API to support image I/O. This Package javax.imageio package contains the basic classes and interfaces for describing the contents of image files, for controlling the image reading process (ImageReader), and image writing process (ImageWriter).
Design Constraint

There is no design constraint of the Online DPL project.

1.7 Organization of the Thesis

The thesis portion of the project was divided into five chapters. Chapter One contains the (SRS) software requirement Specification of DPL, purpose of the project, and definitions of terms. Chapter Two describes the architecture and detailed design of DPL. Chapter Three shows testing of DPL GUI. Chapter Four is the maintenance manual for DPL. Chapter Five discusses the conclusion of this project and future directions. The Appendices containing the project follows Chapter Five. Finally, the references for the project are presented.
CHAPTER TWO
SOFTWARE DESIGN

2.1 Design Pattern Library Design

There are 7 classes in the architecture of Online DPL, see Figure 2.1. GUI0 class is the main class for the GUI. WindowCloser0 class which handles the event when a user closes a window. ToolButton0 class handles the event when a user wants to do the drawing. DrawingCanvas0 class does the real drawing job.

DescriptionPanel0 class holds the entire screen. MouseMotionLister0 class handles the mouse motion event. MouseAdapt0 class handles the events if the mouse is pressed or released.

DrawingCanvas0 class and ToolButton0 class are both inner classes inside GUI0 class. These inner classes create an object of a specific class which was defined as private data field inside the scope of their outer classes which are DrawingCanvas0 class and ToolButton0 class. MouseMotionLister0 class and MouseAdapt0 class are inner classes of DrawingCanvas0 class for the same reason. In the constructor of DrawingCanvas0 class, the instances of MouseMotionLister0 class and MouseAdapt0 class are created to add mouse motion listener.
Figure 17. Class Diagram of Design Pattern Library

DescriptionPanel0 class is defined as a private attribute of GUI0 class. An instance of DescriptionPanel0 class was created in GUI0 class by calling its constructor. GUI0 class relies on WindowCloser0 class to handle the event if the window is closed by a user. An instance of WindowCloser0 class was created inside the constructor of GUI0 class. The only method is WindowClosing() which handles the event when the user closes the window.

In ToolBox0 class, an instance of DrawingCanvas0 class is passed as a parameter to the constructor of DrawingCanvas0 class to specify that it is the calling class, which the user is drawing on the canvas. All the drawxxx() methods in DrawingCanvas0 class returns a DrawableInterface Object so
that it can be stored in the drawing canvas later. All the methods in DrawableInterface class are abstract.

2.2 Detailed Design

**GUI0 Class**

GUI0 class is the main class of DPL project. Its attributes include JLabel, JFileChooser (for File dialog box), JTextArea, JButton (to create instances of Button), and JToolBar. Figure 18 shows the class diagram of GUI0 class.

Methods of GUI0 class:

1. ActionPerformed() handles the action Performed when the user clicks menu or buttons, each performed action is an object. If else statements are used to retrieve a diagram of specific pattern. If user clicks Factory Class Diagram menu, then click "Retrieve Factory Class Diagram" submenu, an image will be retrieved and DPL will display the figure in the content panel. All the patterns provided in DPL can be retrieved. For text file retrieval, such as DisCo specification, open (file) will be called for text file retrieval. Each action performed by the user is an event object, and will be handled by the if statement. If an expression is true then a specific file will be opened.
2. `setBar()` method, each tool button on the toolbar is created by calling the ToolButton class. Images on the tool button are created by calling ImageIcon class which is defined in Swing package in java. ActionListerner is added for each button object to listen to the user’s action. All the tool buttons are added to toolbar.

3. In `open()` method, `showOpenDialog()` method of jFileChooser class is called when a user try a open a file without the specified file instance. `getSelectedFile()` is called after user have chosen a file.
4. In open(File file) method, an object of BufferedInputStream class is created by calling its constructor with an object of FileInputStream class as its argument. Buffered streams read a chunk of bytes at one from a file and append them to a text field to display.

5. In save() method, showOpenDialog() method of JFileChooser class is called when a user try to save a file without the specified file instance. getSelectedFile() is called after user have chosen a file.

6. In save(File file) method, an object BufferedOutputStream class created by calling its constructor with an object of FileOutputStream class as its argument. Buffered streams are read from text field as a chunk of bytes at one and written to buffered streams.

7. In copy() method, highlight text by user is coped to clipboard.

8. In paste() method, text on system clipboard is pasted to text field.

9. In textValueChanged() method, if text have been modified, this function will be notified.

10. lostOwnership() method is an abstract function to notify an object if it is no longer the owner of the contents of the clipboard.

11. In itemStateChanged() method, if an item has been changed, call setDisplay() to change the item.
12. `setDisplay()` calls `setMethods()` of `DescriptionPanel` object to set the title, image, text description.

13. In `LoadList()` method, a string array is loaded into the list.

14. In `addPanel()` method, component can be added into the panel.

15. In `saveImage()` method, encodes buffer image as a JPEG data stream, and write the jpeg to a file using an object `FileOutputStream` class as an argument.

16. In `getFileName()` method, an object of `FileDialog` class is created, `getDirectory()` is called and a file name is returned.

**DescriptionPanel10 Class**

`DescriptionPanel` class displays the text area in the image panel. Figure 19 shows the class diagram of `DescriptionPanel10` class.

In `DescriptionPanel10` class, there are three Attributes: Image, Title, aText. There is a constructor in `DescriptionPanel10` class in order to create new panel and add image into the panel.
Methods of DescriptionPanel0 class:

1. setTitle(String title) sets the title of design patterns.

2. setIcon(ImageIcon icon) displays the images of class diagram of design patterns.

3. setTextDescription(String text) displays the details of design patterns.

Drawingcanvas0 Class

There are two inner classes in DrawingCanvas0 class. MouseMotionListener0 class handles the mouse motion event. MouseAdapt0 class handle the mouse pressed and released Event. Figure 20 shows the class diagram of DrawingCanvas0 Class.
Methods of DrawingCanvasO class:

1. In addObj() method, if a user click a point on canvas to start drawing, a starting point will be stored in an array. When user release the mouse from dragging to finish his drawing job, the ending point will be stored in an array. Also, then object of type which user is drawing will be determined. If numbers of current object does not exceed maximum number of objects can be drawn on canvas, those objects will be added.
2. `drawClass()` method draws a class by calling `drawRect()` and `drawLine()` in Graphics class then return a `DrawableInterface` object.

3. `drawAggregation()` draw a class by calling `drawLine()` in Graphics class then return a return a `DrawableInterface` object.

4. `DrawableInterface` draw a class by calling `drawLine()` in Graphics class to draw the Inheritance triangle then return a `DrawableInterface` object.

5. `DrawLine()` draws a line then return a in `DrawableInterface` object.

6. `DrawText()` draws a string by calling `drawString()` in Graphics class to draw the Inheritance triangle then return a `DrawableInterface` object.

7. `paint()` creates a graphic object, then converts it to a `Graphic2D` object. It also creates off screen image by creating an instance of `BufferedImage` class. It calls `CreateGraphics()` method of `BufferedImage` class to draw an image into the memory buffer. As long as user is drawing, switch among the cases of the Object’s type which user is drawing and perform the drawing actions.

8. In `redraw()` method, If a user clicks “Reset All” button, resets all the x, y coordinates to 0, and clears the canvas.

Methods of `MouseAdapt2` class:
1. `mousePressed()` handles the event when a user press the mouse and start drawing, gets the position of mouse clicked.

2. `mouseReleased()` handles the event when a user release the mouse to stop drawing, add an object.

Methods of `MouseMotionListener` class:

- `mouseDragged()` handles the event when a user drag the mouse for drawing, listen to the mouse dragged action, repaint if necessary.

**DrawableInterface Class**

DrawableInterface is a interface that defines two abstract methods: `addToCanvas()` and `draw()`. Figure 21 shows the class diagram of DrawableInterface.

Methods of DrawableInterface class:

1. `addToCanvas()` is for adding object to canvas.

2. `draw()` does the drawing. The only parameter is a Graphic object.

![Figure 21. Class Diagram of DrawableInterface](image)
ToolBox0 Class

ToolBox0 class has an action() to handle the action performed by user who want to do what kind of drawing. It also defines all the button attributes. Figure 22 shows the class diagram of ToolBox0 class. A constructor takes an instance of DrawingCanvas0 class as a calling object.

Methods of ToolBox0 class:

1. Action() handle the action when user click a button on the right of the GUI. This function has two parameters, an action event object and an integer indicates the drawing object’s type.

2. A constructor which takes an object of DrawingCanvas class as a parameter.

<table>
<thead>
<tr>
<th>ToolBox0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(from default)</td>
</tr>
<tr>
<td>Aggregation: JButton</td>
</tr>
<tr>
<td>C: DrawingCanvasO</td>
</tr>
<tr>
<td>ClassRect: JButton</td>
</tr>
<tr>
<td>DeleteLast: JButton</td>
</tr>
<tr>
<td>Inheritance: JButton</td>
</tr>
<tr>
<td>Line: JButton</td>
</tr>
<tr>
<td>Reset: JButton</td>
</tr>
<tr>
<td>Text: JButton</td>
</tr>
<tr>
<td>+ToolBoxO(default.DrawingCanvasO)</td>
</tr>
<tr>
<td>Action(default.ActionEvent, int)</td>
</tr>
</tbody>
</table>

Figure 22. Class Diagram of ToolBox0 Class
**Toolbutton Class**

ToolButton class defines all tool buttons on toolbar. A constructor with two parameters, string displays on the button and the image on the button. Figure 23 shows the class diagram of ToolButton class.

Methods of ToolButton class:

A constructor contains two parameters: a string object to display string on the button and an image icon object to display the image on the button.

```
ToolButton
  (from default)
+ToolButton(String, Icon)
```

Figure 23. Class Diagram of ToolButton

**Windowcloser0 Class**

WindowCloser0 class defines the window closing event clicked by user. Figure 24 shows the class diagram of WindowCloser0 class.

Methods of WindowCloser0 class:

WindowClosing() methods takes a WindowEvent object as a parameter. System.exit() is called if user close the window. Figure 24 shows class diagram of WindowCloser0 class.
Figure 24. Class Diagram of WindowCloser0 Class
CHAPTER THREE
SOFTWARE QUALITY ASSURANCE

3.1 Introduction

Chapter Three documents the software quality assurance. Specifically, test cases are shown to functionalities of Online DPL. Screen shots show the test cases that were used.

3.2 Test Plans

To demonstrate the functionalities of Online DPL within the scope of SRS[7], the following test cases are described. The first is a scenario of how a user can browse the content of the Online DPL. The second is a scenario of how a user can create/modify a design pattern and then store the design patterns into its own local directory.

Figure 25 shows the GUI (Graphical User Interface) of DPL when a client opens the application for the first time.
To open a text file, click on "Open" button in toolbar. An "Open" file dialog will displayed for the user to choose a text file. The snapshot of this GUI (graphic user interface) is shown Figure 26.

Figure 26. External Interface
Figure 26. Open a Text File

Figure 27 shows a text file, with DisCo specification, was opened.
The Combination of Observer and Mediator in DisCo specification.

```plaintext
class Manager = Mediator.
class MgdSubject = Subject + Colleague.
class MgdObserver = Observer + Colleague.
MgdConnect(mgr: Manager, mb: Mbox; ms: MgdSubject):
refines Connect(m: Mediator = mgr; mb.Mbox = mb; c: Colleague = ms)
for m belong to Manager, c belong to MgdSubject.

Register(ms: MgdSubject; mo: MgdObserver; mb: Mbox; mgr Manager):
refines Attach(s: Subject = ms: Observer = mo)
for s belongs to MgdSubject, o belongs to MgdObserver
```

Figure 27. Opened Text File (Distributed Cooperation Specification)

After the user has modified some text in DisCo specification, he/she can click the “Save” button in the toolbar to save the text file. Figure 28 shows the save file dialog.
The Combination of Observer and Mediator in DIsCo specification.

```java
class Manager = Mediator;
class MgdSubject = Subject + Collc
class MgdObserver = Observer + C
MgdConnect(mgr:Manager, mb: Mb;
refines Connect(m;Mediator = mgr,
for m belong to Manager, c belong t
Register(ms: MgdSubject; mc: Mgd
```

Figure 28. Save File Dialog

When a user wants to browse a specific design pattern, He/she should click on the menu, then choose a pattern from the submenu. For example, observer pattern is under the catalog of behavior pattern, so the user will click on behavior menu, then choose “Retrieve Observer Pattern”. The user chooses which diagram he/she wants to see from the Observer submenu. A diagram will be displayed after the user clicks on the submenu. Figure 29 shows the window after a user clicks on “Observer Sequence Diagram”. 
Figure 29. Submenu Showing Observer Sequence Diagram

Figure 30 shows the Observer design pattern's sequence diagram being opened. It shows the interaction among concreteObserver object and concreteSubject object.
Figure 30. Opened Observer Pattern Sequence Diagram

Figure 31 shows a user has chosen to open an observer pattern class diagram. ChangeManager class minimizes the work required to make a design pattern reflect a change in their content. If an operation involves changes to several interdependent subjects, ensure that their observers are notified only after all the subjects have been modified to avoid notifying observers more than once.
Figure 31. An Observer Design Pattern Class Diagram is Opened

Figure 32 shows the drawing functionalities of Online DPL. Click on “class” button to dynamically draw a class, click on “Inheritance” button to draw Inheritance triangle, click on “Text” button for text input, and click “Aggregation” for Aggregation relationship between classes.
Figure 32. The Snapshot of Drawing Functionality

Figure 33 to Figure 37 shows a sequence of steps on how a mediator pattern is drawn.

In Figure 33 the user inputs text in the text field at the bottom of the GUI and presses “Modify Class” button to store the text he/she just typed in, then the user clicks on the “Text” button to draw the text on the drawing canvas where a user can display the class name, method, or attribute.
Figure 33. User Inputs Text "Mediator" and "ConcreteMediator"

Figure 34 shows the user inputs a new class name "ConcreteColleague" in the text field at the bottom of the GUI. Mediator, Colleague, and ConcreteColleague are all class names in mediator design pattern.
Figure 34. User Types “Colleague”

Figure 35 shows the user types in more class names, which are “ConcreteColleague1” and “ConcreteColleague2”. Mediator is the superclass of ConcreteMediator class. Colleague class is the superclass of both ConcreteColleague1 and ConcreteColleague2 classes.
Figure 35. User Types "ConcreteColleague1" and "ConcreteColleague2"

Figure 36 shows the user clicks on the "Class" button at the right of the GUI and then clicks the mouse at the location where he/she wants to draw the class shape, the user then releases the mouse after finish to draw each class object.
If a user wants to put inheritance between class boxes, he/she will click on the "Inheritance" button at the right of the GUI then clicks on the canvas. Figure 37 shows the inheritance symbol was drawn. To draw a line, the user clicks on the "Line" button, at the right of the GUI then clicks on the canvas.
When the user wants to reload an image, he/she clicks on "Reload Image" Button on the toolbar. Figure 38 shows an open image file dialog.
Figure 38. Reload Image File Dialog

Figure 39 shows a Mediator pattern class diagram after the user has chosen it. This class diagram was drawn before by the user, now it is retrieved for modification. A new dialog is opened along with the image waiting for user's input in text field to modify the class diagram. The user can input attribute and method of Mediator. The user can input the x and y coordinate to indicate where the Attribute and Method will be drawn on the image. "Gap" button allows the user to input the differences of y coordinate between Attribute and Method. "Modify" button allows the user to modify the class diagram.
“Save Image” button allows the user to store the image back to local file directories.

Figure 39. A Reloaded Mediator Pattern Class

Figure 40 shows the user types in attribute named “Attribute2” and clicks on attribute button, types in method named “Method2” and clicks on Method button along with the x and y coordinate where the attribute and method will be drawn. The user clicks “Modify” button to modify the class diagram.
Figure 40. User Types in Attribute and Click on Attribute Button to Store the Attribute and Do the Same for the Method

Figure 41 shows the class diagram of Mediator pattern was modified.
Figure 42. The User Inputs More Attributes and Methods and Then Clicks on the "Modify" Button

In Figure 43 shows the user click on the "Save Image" button, a save file dialog is opened for the user to type in a file name. The user types in a file name as "Test2.jpg".
Figure 43. Save File Dialog is Opened After the User Click the "Save Image" Button
CHAPTER FOUR

MAINTENANCE

4.1 Introduction

Included in Chapter Four was a presentation of the maintenance manual of the project.

4.2 File Structure

In the DPL project, there are 6 java files. All the source files are stored in the attached CD. All java files are put in directory /java. The directory /document all word files are stored.

Here is the list of source files:

- GUI.java (ToolBox class and DrawingCanvas class are inner class in GUI class)
- DescriptionPanel.java
- SaveImageTest.java
- ToolButton.java
- WindowCloser.java
- DrawableInterface.java

Here is the list of bytes code:

- GUI.java.class
- DescriptionPanel.class
- SaveImageTest.class
- ToolButton.class
- WindowCloser.class
- DrawableInterface.class

Here is the list of bytes code:

- Welcome.html
- Tutorial.html

4.3 Recompile

The source code can be recompiled with Jbuilder 4 or older version. If the user does not have Jbuilder compiler, he/she need to set up the following environment variables for running java programs in the DOS environment in Window:

Set path=%path%;c:\jdk1.4\bin
Set classpath=;%classpath%
C:\jdk1.4 is the directory where Java development kit in installed.

To compile the file, type:

Javac filename.java

To run the program, type:

Java classname

The user needs to look up all the classes dependency for the main program in order to get the expected output. All the dependent classes should be stored in the proper class path.
4.4 Installation Process

A user can install all the java files from the attached CD to the directory he wants to compile or run the application.

A user may install DPL from a DPL.ZIP file

1. unzip DPL.ZIP with a Win-ZIP program.
2. extract files and install to a new directory named DPL.
3. after installation, there should be subdirectories such as DOC which contains the documentation files, Image directory which holds all the image files in DPL. bin directory contains executable files such as DPL.exe. Java directory contains all the java source code in DPL. Class directory contains all the bytes code in DPL. Welcome.html is not in any directory.
4. double click the icon of DPL.exe to call the GUI application.

More design patterns (such as visitor, command, template, strategy design patterns) can be added into the Online DPL.

4.5 Trouble Shooting

If there is any problem, the user should look up the trouble shooting page or FAQ of Online DPL.
CHAPTER FIVE
CONCLUSIONS AND FUTURE DIRECTIONS

5.1 Summary of Results

The online DPL project holds the specifications of six generic and one domain-specific design patterns. This is the first tool developed for the ADAP model that allows software engineers to create their own design patterns, and modify and store it back to their own local file directories.

Online DPL allows the software engineer to browse different specifications of a given set of design patterns. Drawing capabilities are implemented so not only can a user create a class diagram, but can also draw the relationship among classes. Without the drawing capabilities, all of the images of the design patterns in the CD, or Website cannot be referred and modified. The significance of the Online DPL project is that it provides a framework for ADAP model and also a tool for software engineers to learn design patterns, and to create new design patterns other than the 23 generic design patterns that are found in Gamma [2].

Software engineers can benefit from the Online DPL. DPL allows a user to dynamically create his/her own design patterns (a user can input texts such as class name, attributes, and methods from text field), save the design patterns created, retrieve the images (class diagrams) stored

65
before, then modify and store them back to the user’s local file directories.

Nowadays, the study of design patterns is important, but there is no such tool like Online DPL that allows users to create, modify and store design patterns. Existing on-line design patterns Websites only provides either the resources such as related Websites or explanation of design patterns for research. These Web pages only allow users to browse the design patterns stored in that site. None of them offer an application like DPL where users can create their own design patterns, modify and store them in the users’ local file directories. After finishing this project, I have learned to create GUI (Graphical User Interface), Java 2D Graphics, and learn how to create, modify and store images. Also, I learned JSP (Java Server Pages). During the research of design patterns, I learned the benefit of using design patterns in designing software products and realized why the study of design patterns is important.

5.2 Future Directions

First, add more generic or domain-specific design patterns to the basic set of DPL. Since DPL only contains six generic design patterns and one domain-specific design patterns, more design patterns such as factory method design patterns, prototype design patterns, singleton design patterns in creational design patterns catalog can be added, structural
design patterns such as bridge, adaptor, decorator, façade, proxy design patterns can also be added to DPL. Domain-specific design pattern such as Master-Slave searcher pattern can be added into DPL. In this design pattern, the master dispatches multiple aglets to search the remote hosts. The slave aglets will visit all the hosts, combine the results and send them back to the host.

Second, providing a capability to combine or integrate existing design patterns to create a new design pattern so that new epic of design patterns can be created or discovered. Third, include a way to reverse engineer of an existing framework or application to detect the underlying design patterns that have been used so that software engineers can get the generic ideas on how design patterns can be combined, modified and created.

Fourth, the automatic generation of class diagrams from a specifications can be used to store any kind of structured information, to create the class diagrams.

Fifth, automatic source code generation from the class diagrams of a specific design patterns can be a next step.

Lastly, development of a verification tool to do analysis of specification to determine correctness, incompleteness, and consistency of a newly created design pattern.
APPENDIX A

JAVA SOURCE CODE
//GUI0.java
//drawing application
// Save and Reload images
import java.io.*;
import com.sun.image.codec.jpeg.*;
import java.util.Enumeration;
import java.net.URL;
import java.sql.*;
import java.util.*;
import javax.swing.*;
import javax.swing.text.*;
import javax.swing.event.*;
import javax.swing.border.*;
import java.awt.geom.*;
import java.awt.*;
import java.awt.Canvas;
import java.awt.Canvas.*;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.*;
import java.awt.datatransfer.*;
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.Color;
import java.awt.Font;
import java.awt.geom.*;
import java.awt.image.BufferedImage;
public class GUI0 extends JFrame implements
ActionListener, ItemListener,
TextListener, ClipboardOwner
{
private BufferedImage expImage;

private int width, height;
private Image image;
private BufferedImage bufferImage;
private MediaTracker mediaTracker;
private JLabel jl = new JLabel();
private JFileChooser fc;
static final String nl = System.getProperty("line.separator");

private JLabel jl = new JLabel();
private JFileChooser jFileChooser = new JFileChooser();
private JTextArea theText = new JTextArea();
private JTextField DataInput = new JTextField(20);
private String[] content = new String[20];
private int stringCount = 0;
private String factory = "No such pattern.";
private String builder = "No such pattern.";
private String com = "No such pattern.";
private String fly = "No such pattern.";
private String observer = "No such pattern.";
private String mediator = "No such pattern.";
private String ms = "No such pattern.";


// Declare an ImageIcon array
private ImageIcon[] Image = new ImageIcon[9];

private String[] Description = new String[9];

// Declare and create a description panel
private DescriptionPanel0 descriptionPanel = new DescriptionPanel0();

//private boolean modified = false;
private JComboBox c;
//private static final int MAXBJECTS = 20;
private DrawableInterface[ ] Object = new DrawableInterface[20];

private int numObjects = 0;
private DrawableInterface selected;
private DrawingCanvas0 canvas = new DrawingCanvas0();
private LoadImageTest tr ;
private SaveImgTest0 s ;
//Database db;
//TextArea query;
JButton Clear, Save, Quit, Copy, Cut, Paste, Home, Open, QS, Preview;
JButton StoreObj, RestoreObj, Input;
JToolBar toolbar;

//java.awt.List Tables, Columns, Data;
ToolBox0 ToolPanel;
public static void main(String[] args) {
GUI0 g = new GUI0();
g.setBackground(Color.white);
g.setSize(800,550);
g.setVisible(true);
}
public GUI0() // constructor
{   //content[0] = "ClassName";
content[0] = "";
WindowListener wl = new WindowCloser0();
addWindowListener(wl);
setSize(WIDTH, HEIGHT);
addWindowListener(new WindowDestroyer());
setTitle("GUI To Store and Retrieve Pattern GUI, Drawing Application");
Container contentPane = getContentPane();
JPanel tool = new JPanel(); // create a panel named tool
tool.setLayout(new BorderLayout());
toolbar = new JToolBar();
tool.add("North", toolbar);

setBar(); // set tollButton bar
// set tooBar, Panel tool hold the tool buttons
g getContentPane().add(tool, BorderLayout.NORTH);

DrawingCanvasO canvas = new DrawingCanvasO();

ToolPanel = new ToolBoxO(canvas);
getContentPane().add(ToolPanel, BorderLayout.EAST);
getContentPane().add(new JScrollPane(theText),
BorderLayout.WEST);

getContentPane().add(canvas, BorderLayout.CENTER);
setVisible(true);

JPanel p1 = new JPanel();
p1.setLayout(new BorderLayout());
p1.add(new JLabel("Change Class, Attribute, Methods"),
BorderLayout.CENTER);
p1.add( DataInput, BorderLayout.CENTER);
Input = new JButton("Modify Class");
Input.addActionListener(this);
p1.add( Input, BorderLayout.EAST);
getContentPane().add(p1, BorderLayout.SOUTH);

// Load images info Image array
Image[2] = new ImageIcon("factory.jpg");
Image[3] = new ImageIcon("builder.jpg");
Image[5] = new ImageIcon("flyweight.jpg");
Image[6] = new ImageIcon("meeting.jpg");
Image[7] = new ImageIcon("mgdObs.jpg");

// Set text description
Description[2] = "Object interaction of Factory Pattern";
Description[3] = "Object interaction of Builder Pattern";
Description[4] = "Object interaction of Composite Pattern";
Description[5] = "Object interaction of FlyWeight Pattern";
Description[6] = "Object interaction of Meeting Pattern";
Description[7] = "Object interaction of Managed Observer Pattern";

// Description[8] = "Object interaction of Visitor Pattern";

// Create items into the combo box
C = new JComboBox(Title);

// Set default directory to the current directory
jFileChooser.setCurrentDirectory(new File("."));

// Register listener
C.addItemListener(this);

// creational patterns
// create JMenu object CMenu
JMenu CMenu = new JMenu("Creational");
JMenuItem cm;

JMenu sFacMenu = new JMenu("Store Factory Pattern");
sFacMenu.addActionListener(this);
CMenu.add(sFacMenu);

/************************************************************/
// Create SubMenu
JMenuItem mfs;
mfs = new JMenuItem("Store Factory Use Case Diagram");
mfs.addActionListener(this);
sFacMenu.add(mfs);
mfs = new JMenuItem("Store Factory Class Diagram");
mfs.addActionListener(this);
sFacMenu.add(mfs);
mfs = new JMenuItem("Store Factory Sequence Diagram");
mfs.addActionListener(this);
sFacMenu.add(mfs);
mfs = new JMenuItem("Store Factory DisCo Specification");
mfs.addActionListener(this);
sFacMenu.add(mfs);

/********************************************/

cm = new JMenuItem("Store Builder Pattern");
cm.addActionListener(this);
CMenu.add(cm);
JMenu FacMenu = new JMenu("Retrieve Factory Pattern");
FacMenu.addActionListener(this);
CMenu.add(FacMenu);

/********************************************/
// Create SubMenu
JMenuItem mf;
mf = new JMenuItem("Factory Use Case Diagram");
mf.addActionListener(this);
FacMenu.add(mf);
mf = new JMenuItem("Factory Class Diagram");
mf.addActionListener(this);
FacMenu.add(mf);
mf = new JMenuItem("Factory Sequence Diagram");
mf.addActionListener(this);
FacMenu.add(mf);
mf = new JMenuItem("Factory DisCo Specification");
mf.addActionListener(this);
FacMenu.add(mf);

/********************************************/

cm = new JMenuItem("Retrieve Builder Pattern");
cm.addActionListener(this);
CMenu.add(cm);
cm = new JMenuItem("Clear");
cm.addActionListener(this);
CMenu.add(cm);
cm = new JMenuItem("Exit");
cm.addActionListener(this);
CMenu.add(cm);

// Structural patterns
// create JMenu object SMenu
JMenu SMenu = new JMenu("Structural");
JMenuItem m;
JMenu scomMenu = new JMenu("Store Composite Pattern");
scomMenu.addActionListener(this);
SMenu.add(scomMenu);

/*********************************************/
// Create SubMenu
JMenuItem scom;
scom = new JMenuItem("Store Composite Use Case Diagram");
scom.addActionListener(this);
scomMenu.add(scom);
scom = new JMenuItem("Store Composite Class Diagram");
scom.addActionListener(this);
scomMenu.add(scom);
scom = new JMenuItem("Store Composite Sequence Diagram");
scom.addActionListener(this);
scomMenu.add(scom);
scom = new JMenuItem("Store Composite DisCo Specification");
scom.addActionListener(this);
scomMenu.add(scom);

/*********************************************/

JMenu sflyMenu = new JMenu("Store FlyWeight Pattern");
sflyMenu.addActionListener(this);
SMenu.add(sflyMenu);
Create SubMenu

JMenuItem sfm;
sfm = new JMenuItem("Store FlyWeight Use Case Diagram");
sfm.addActionListener(this);
sflyMenu.add(sfm);
sfm = new JMenuItem("Store FlyWeight Class Diagram");
sfm.addActionListener(this);
sflyMenu.add(sfm);
sfm = new JMenuItem("Store FlyWeight Sequence Diagram");
sfm.addActionListener(this);
sflyMenu.add(sfm);
sfm = new JMenuItem("Store FlyWeighty DisCo Specification");
sfm.addActionListener(this);
sflyMenu.add(sfm);

JMenu comMenu = new JMenu("Retrieve Composite Pattern");
comMenu.addActionListener(this);
SMenu.add(comMenu);

// Create SubMenu

JMenuItem com;
com = new JMenuItem("Composite Use Case Diagram");
com.addActionListener(this);
comMenu.add(com);
com = new JMenuItem("Composite Class Diagram");
com.addActionListener(this);
comMenu.add(com);
com = new JMenuItem("Composite Sequence Diagram");
com.addActionListener(this);
comMenu.add(com);
com = new JMenuItem("Composite DisCo Specification");
com.addActionListener(this);
comMenu.add(com);

/********************
JMenu flyMenu = new JMenu("Retrieve FlyWeight Pattern");
flyMenu.addActionListener(this);
SMenu.add(flyMenu);

/********************
// Create SubMenu
JMenuItem fm;
fm = new JMenuItem("FlyWeight Use Case Diagram");
fm.addActionListener(this);
flyMenu.add(fm);
fm = new JMenuItem("FlyWeight Class Diagram");
fm.addActionListener(this);
flyMenu.add(fm);
fm = new JMenuItem("FlyWeight Sequence Diagram");
fm.addActionListener(this);
flyMenu.add(fm);
fm = new JMenuItem("FlyWeighty DisCo Specification");
fm.addActionListener(this);
flyMenu.add(fm);

/********************
fm = new JMenuItem("Clear");
m.addActionListener(this);
SMenu.add(m);
fm = new JMenuItem("Exit");
m.addActionListener(this);
m.add(fm);

// Behavioral patterns
// create JMenu object BMenu
JMenu BMenu = new JMenu("Behavioral");
JMenuItem bm;
JMenu sObsMenu = new JMenu("Store Observer Pattern");
sObsMenu.addActionListener(this);
BMenu.add(sObsMenu);

/**********************************
  // Create SubMenu
  JMenuItem smo;
  smo = new JMenuItem("Store Observer Use Case Diagram");
  smo.addActionListener(this);
  sObsMenu.add(smo);
  smo = new JMenuItem("Store Observer Class Diagram");
  smo.addActionListener(this);
  sObsMenu.add(smo);
  smo = new JMenuItem("Store Observer Sequence Diagram");
  smo.addActionListener(this);
  sObsMenu.add(smo);
  smo = new JMenuItem("Store Observer DisCo Specification");
  smo.addActionListener(this);
  sObsMenu.add(smo);
  JMenu smedMenu = new JMenu("Store Mediator Pattern");
  smedMenu.addActionListener(this);
  BMenu.add(smedMenu);

/**********************************
  // Create SubMenu
  JMenuItem sme;
  sme = new JMenuItem("Store Mediator Use Case Diagram");
  sme.addActionListener(this);
  smedMenu.add(sme);
  sme = new JMenuItem("Store Mediator Class Diagram");
  sme.addActionListener(this);
  smedMenu.add(sme);
  sme = new JMenuItem("Store Mediator Sequence Diagram");
  sme.addActionListener(this);
  smedMenu.add(sme);
  sme = new JMenuItem("Store Mediator DisCo Specification");
  sme.addActionListener(this);
  smedMenu.add(sme);
smedMenu.add(sme);

/**************************************************
JMenu ObsMenu = new JMenu("Retrieve Observer Pattern");
ObsMenu.addActionListener(this);
BMenu.add(ObsMenu);

/**************************************************/
// Create SubMenu
JMenuItem mo;
mo = new JMenuItem("Observer Use Case Diagram");
mo.addActionListener(this);
ObsMenu.add(mo);
mo = new JMenuItem("Observer Class Diagram");
mo.addActionListener(this);
ObsMenu.add(mo);
mo = new JMenuItem("Observer Sequence Diagram");
mo.addActionListener(this);
ObsMenu.add(mo);
mo = new JMenuItem("Observer DisCo Specification");
mo.addActionListener(this);
ObsMenu.add(mo);
JMenu medMenu = new JMenu("Retrieve Mediator Pattern");
medMenu.addActionListener(this);
BMenu.add(medMenu);

/**************************************************/
// Create SubMenu
JMenuItem me;
me = new JMenuItem("Mediator Use Case Diagram");
me.addActionListener(this);
medMenu.add(me);
me = new JMenuItem("Mediator Class Diagram");
me.addActionListener(this);
medMenu.add(me);
me = new JMenuItem("Mediator Sequence Diagram");
me.addActionListener(this);
medMenu.add(me);
me = new JMenuItem("Mediator DisCo Specification");
me.addActionListener(this);
medMenu.add(me);

/*************************************************************/
bm = new JMenuItem("Clear");
bm.addActionListener(this);
BMenu.add(bm);
bm = new JMenuItem("Exit");
bm.addActionListener(this);
BMenu.add(bm);

// Domain-Specific Pattern
JMenu DMenu = new JMenu("Domain-Specific");
JMenuItem dm;
JMenu smsMenu = new JMenu("Store Meeting Pattern");
smsMenu.addActionListener(this);
DMenu.add(smsMenu);

/*************************************************************/
// Create SubMenu
JMenuItem sms;
 sms = new JMenuItem("Store Meeting Use Case Diagram");
 sms.addActionListener(this);
 smsMenu.add(sms);
 sms = new JMenuItem("Store Meeting Class Diagram");
 sms.addActionListener(this);
 smsMenu.add(sms);
 sms = new JMenuItem("Store Meeting Sequence Diagram");
 sms.addActionListener(this);
 smsMenu.add(sms);
 sms = new JMenuItem("Store Meeting DisCo Specification");
 sms.addActionListener(this);
 smsMenu.add(sms);

80
JMenu msMenu = new JMenu("Retrieve Meeting Pattern");
msMenu.addActionListener(this);
DMenu.add(msMenu);

// Create SubMenu
JMenuItem ms;
ms = new JMenuItem("Meeting Use Case Diagram");
ms.addActionListener(this);
msMenu.add(ms);
ms = new JMenuItem("Meeting Class Diagram");
ms.addActionListener(this);
msMenu.add(ms);
ms = new JMenuItem("Meeting Sequence Diagram");
ms.addActionListener(this);
msMenu.add(ms);
ms = new JMenuItem("Meeting DisCo Specification");
ms.addActionListener(this);
msMenu.add(ms);

dm = new JMenuItem("Clear");
dm.addActionListener(this);
DMenu.add(dm);
dm = new JMenuItem("Exit");
dm.addActionListener(this);
DMenu.add(dm);

// combination
JMenu combMenu = new JMenu("Combination");
JMenuItem comb;
JMenu smgdMenu = new JMenu("Store Managed Observer Pattern");
smgdMenu.addActionListener(this);
combMenu.add(smgdMenu);
I Create SubMenu

JMenuItem smb;
smb = new JMenuItem("Store Managed Observer Use Case Diagram");
smb.addActionListener(this);
smgdMenu.add(smb);
smb = new JMenuItem("Store Managed Observer Class Diagram");
smb.addActionListener(this);
smgdMenu.add(smb);
smb = new JMenuItem("Store Managed Observer Sequence Diagram");
smb.addActionListener(this);
smgdMenu.add(smb);

/*****************************/
JMenu mgdMenu = new JMenu("Retrieve Managed Observer Pattern");
mgdMenu.addActionListener(this);
combMenu.add(mgdMenu);

/*****************************/
// Create SubMenu
JMenuItem mb;
mb = new JMenuItem("Managed Observer Use Case Diagram");
mb.addActionListener(this);
mgdMenu.add(mb);
mb = new JMenuItem("Managed Observer Class Diagram");
mb.addActionListener(this);
mgdMenu.add(mb);
mb = new JMenuItem("Managed Observer Sequence Diagram");
mb.addActionListener(this);
mgdMenu.add(mb);
mb = new JMenuItem("Managed Observer DisCo Specification");
mb.addActionListener(this);
mgdMenu.add(mb);
/
**************************
comb = new JMenuItem("Multiple Inheritance");
comb.addActionListener(this);
combMenu.add(comb);

// Reverse
JMenu RMenu = new JMenu("Reverse Engineering");
JMenuItem rev;
rev = new JMenuItem("Detect Underlying Pattern");
rev.addActionListener(this);
RMenu.add(rev);
rev = new JMenuItem("Mobile Agent");
rev.addActionListener(this);
RMenu.add(rev);

// about
JMenu abMenu = new JMenu("About");
JMenuItem ja;
ja = new JMenuItem("About Design Pattern");
ja.addActionListener(this);
abMenu.add(ja);
ja = new JMenuItem("About Design Pattern Library");
ja.addActionListener(this);
abMenu.add(ja);
ja = new JMenuItem("About DisCo");
ja.addActionListener(this);
abMenu.add(ja);
ja = new JMenuItem("Purpose of this Project");
ja.addActionListener(this);
abMenu.add(ja);
// Create MenuBar
JMenuBar mBar = new JMenuBar();

// Add JMenu object to JMenuBar
mBar.add(CMenu);
mBar.add(SMenu);
mBar.add(BMenu);
mBar.add(DMenu);
mBar.add(combMenu);
mBar.add(RMenu);
mBar.add(abMenu);
setJMenuBar(mBar);
}

public void actionPerformed(ActionEvent e) {
String actionCommand = e.getActionCommand();
Object obj = e.getSource();

// Retrieve Patterns
if (actionCommand.equals("Factory Use Case Diagram")) {
Image[2] = new ImageIcon("facUse.jpg");

// text description
Description[2] = "Use Case Diagram of Factory Pattern";
setDisplay(2);
getContentPane().add(descriptionPanel, BorderLayout.WEST);
setVisible(true);
}
else if (actionCommand.equals("Factory Class Diagram")) {
Image[2] = new ImageIcon("facC.gif");
Description[2] = "Class Diagram of Factory Pattern";
setDisplay(2);
getContentPane().add(descriptionPanel, BorderLayout.WEST);
setVisible(true);
}
else if (actionCommand.equals("Factory Sequence Diagram"))
{
    Image[2] = new ImageIcon("facSeq.gif"); // reset image
    Description[2] = "Sequence Diagram of Factory Pattern";
    setDisplay(2);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Factory Object Interaction"))
{
    setDisplay(2);
    Image[2] = new ImageIcon("factory.jpg");
    Description[2] = "Object Interaction of Factory Pattern";
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Factory DisCo Specification"))
{
    File facD = new File("c:\DPL\fac_disco.txt");
    open(facD);
}
else if (actionCommand.equals("Clear"))
{
    theText.setText("*");
}
else if (actionCommand.equals("Get Builder Pattern"))
{
}
else if (actionCommand.equals("Composite Use Case Diagram"))
{
    Image[4] = new ImageIcon("compoUse.jpg");
    Description[4] = "Use Case Diagram of Composite Pattern";
setDisplay(4);
getContentPane().add(descriptionPanel,
BorderLayout.WEST);
setVisible(true);
}
else if (actionCommand.equals("Composite Class Diagram"))
{
Description[4] = "Class Diagram of Composite Pattern";
setDisplay(4);
getContentPane().add(descriptionPanel,
BorderLayout.WEST);
setVisible(true);
}
else if (actionCommand.equals("Composite Sequence Diagram"))
{
Image[4] = new ImageIcon("comSeq.gif"); // reset image
Description[4] = "Sequence Diagram of Composite Pattern";
setDisplay(4);
getContentPane().add(descriptionPanel, BorderLayout.WEST);
setVisible(true);
}
else if (actionCommand.equals("Composite Object Interaction"))
{
setDisplay(4);
Description[4] = "Object Interaction of Composite Pattern";
getContentPane().add(descriptionPanel, BorderLayout.WEST);
setVisible(true);
}
else if (actionCommand.equals("Composite DisCo Specification"))
{
File compoD = new File("c:\DPL\compo_disco.txt");
open(compoD);
}
else if (actionCommand.equals("FlyWeight Use Case Diagram"))
{
    Image[5] = new ImageIcon("flyUse.jpg");
    Description[5] = "Use Case Diagram of FlyWeight Pattern";
    setDisplay(5);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("FlyWeight Class Diagram"))
{
    Description[5] = "Class Diagram of FlyWeight Pattern";
    setDisplay(5);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("FlyWeight Sequence Diagram"))
{
    Image[5] = new ImageIcon("flySeq.gif"); // reset image
    Description[5] = "Sequence Diagram of FlyWeight Pattern";
    setDisplay(5);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("FlyWeight Object Interaction"))
{
    setDisplay(5);
    Image[5] = new ImageIcon("flyweight.jpg");
    Description[5] = "Object Interaction of FlyWeight Pattern";
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("FlyWeight DisCo Specification"))
{
File flyD = new File("c:\DPL\fly_disco.txt");
open(flyD);
}

else if (actionCommand.equals("Observer Use Case Diagram"))
{
    Image[1] = new ImageIcon("obsUse.jpg");
    Description[1] = "Use Case Diagram of Observer Pattern";
    setDisplay(1);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Observer Class Diagram"))
{
    Image[1] = new ImageIcon("obsC.gif");
    Description[1] = "Class Diagram of Observer Pattern";
    setDisplay(1);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Observer Sequence Diagram"))
{
    Image[1] = new ImageIcon("obsSeq.gif"); // reset image
    Description[1] = "Sequence Diagram of Observer Pattern";
    setDisplay(1);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Observer Object Interaction"))
{
    setDisplay(1);
    Image[1] = new ImageIcon("Observer.jpg");
    Description[1] = "Object Interaction of Observer Pattern";
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
else if (actionCommand.equals("Observer DisCo Specification"))
{
    File obsD = new File("c:\DPL\obs_disco.txt");
    open(obsD);
}
else if (actionCommand.equals("Mediator Use Case Diagram"))
{
    Image[0] = new ImageIcon("medUse.jpg");
    Description[0] = "Use Case Diagram of Mediator Pattern";
    setDisplay(0);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Mediator Class Diagram"))
{
    Image[0] = new ImageIcon("medC.gif");
    Description[0] = "Class Diagram of Mediator Pattern";
    setDisplay(0);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Mediator Sequence Diagram"))
{
    Image[0] = new ImageIcon("medSeq.gif"); // reset image
    Description[0] = "Sequence Diagram of Mediator Pattern";
    setDisplay(0);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Mediator Object Interaction"))
{
    Image[0] = new ImageIcon("mediator.jpg");
    Description[0] = "Object interaction of Mediator Pattern";
    setDisplay(0);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
setVisible(true);
}
else if (actionCommand.equals("Mediator DisCo Specification"))
{
    File medD = new File("c:\DPL\med_disco.txt");
    open(medD);
}
else if (actionCommand.equals("Meeting Use Case Diagram"))
{
    Image[6] = new ImageIcon("meetUse.jpg");
    Description[6] = "Use Case Diagram of Meeting Pattern";
    setDisplay(6);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Meeting Class Diagram"))
{
    Image[6] = new ImageIcon("meetC.gif");
    Description[6] = "Class Diagram of Meeting Pattern";
    setDisplay(6);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Meeting Sequence Diagram"))
{
    Image[6] = new ImageIcon("medSeq.gif"); // reset image
    Description[6] = "Sequence Diagram of Meeting Pattern";
    setDisplay(6);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Meeting Object Interaction"))
{
    Image[6] = new ImageIcon("meeting.jpg");
    Description[6] = "Object interaction of Meeting Pattern";
    }
setDisplay(6);
getContentPane().add(descriptionPanel, BorderLayout.WEST);
setVisible(true);
}
else if (actionCommand.equals("Meeting DisCo Specification"))
{
File meetD = new File("c:\DPL\meet_disco.txt");
open(meetD);
}
else if (actionCommand.equals("Managed Observer Use Case Diagram"))
{
Image[7] = new ImageIcon("mgdObsUse.jpg");
Description[7] = "Use Case Diagram of Managed Observer Pattern";
setDisplay(7);
getContentPane().add(descriptionPanel, BorderLayout.WEST);
setVisible(true);
}
else if (actionCommand.equals("Managed Observer Class Diagram"))
{
Image[7] = new ImageIcon("mgdObsC.gif");
Description[7] = "Class Diagram of Managed Observer Pattern";
setDisplay(7);
getContentPane().add(descriptionPanel, BorderLayout.WEST);
setVisible(true);
}
else if (actionCommand.equals("Managed Observer Sequence Diagram"))
{
Image[7] = new ImageIcon("mgdObsSeq.gif"); // reset image
Description[7] = "Sequence Diagram of Managed Observer Pattern";
setDisplay(7);
getContentPane().add(descriptionPanel, BorderLayout.WEST);
setVisible(true);

} else if (actionCommand.equals("Managed Observer Object Interaction"))
{
    Image[7] = new ImageIcon("mgdObs.jpg");
    Description[7] = "Object interaction of Managed Observer Pattern";
    setVisible(true);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Managed Observer DisCo Specification"))
{
    File mgdD = new File("c:\DPL\mgdObs_disco.txt");
    open(mgdD);
}

(/

store Patterns

if (actionCommand.equals("Store Factory Use Case Diagram"))
{
    Image[2] = new ImageIcon("facUse.jpg");
    // text description
    Description[2] = "Use Case Diagram of Factory Pattern";
    setVisible(true);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Store Factory Class Diagram"))
{
    Image[2] = new ImageIcon("facC.gif");
    Description[2] = "Class Diagram of Factory Pattern";
    setVisible(true);
}
getContentPane().add(descriptionPanel, BorderLayout.WEST);
setVisible(true);
}
else if (actionCommand.equals("Store Factory Sequence Diagram"))
{
    Image[2] = new ImageIcon("facSeq.gif"); // reset image
    Description[2] = "Sequence Diagram of Factory Pattern";
    setDisplay(2);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Factory Object Interaction"))
{
    setDisplay(2);
    Image[2] = new ImageIcon("factory.jpg");
    Description[2] = "Object Interaction of Factory Pattern";
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Store Factory DisCo Specification"))
{
    File sfacD = new File("c:\DPL\fac_disco.txt");
    save(sfacD);
}
else if (actionCommand.equals("Get Builder Pattern"))
{
}
else if (actionCommand.equals("Store Composite Use Case Diagram"))
{
    Image[4] = new ImageIcon("compoUse.jpg");
    Description[4] = "Use Case Diagram of Composite Pattern";
    setDisplay(4);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
setVisible(true);
)
else if (actionCommand.equals("Store Composite Class Diagram"))
 {
 Description[4] = "Class Diagram of Composite Pattern";
 setDisplay(4);
 getContentPane().add(descriptionPanel, BorderLayout.WEST);
 setVisible(true);
}
else if (actionCommand.equals("Store Composite Sequence Diagram"))
 {
 Image[4] = new ImageIcon("comSeq.gif"); // reset image
 Description[4] = "Sequence Diagram of Composite Pattern";
 setDisplay(4);
 getContentPane().add(descriptionPanel, BorderLayout.WEST);
 setVisible(true);
}
else if (actionCommand.equals("Composite Object Interaction"))
 {
 setDisplay(4);
 Description[4] = "Object Interaction of Composite Pattern";
 getContentPane().add(descriptionPanel, BorderLayout.WEST);
 setVisible(true);
}
else if (actionCommand.equals("Store Composite DisCo Specification"))
 {
 File scompoD = new File("c:\\DPL\\compo_disco.txt");
 save(scompoD);
}
else if (actionCommand.equals("Store FlyWeight Use Case Diagram"))
{
    Image[5] = new ImageIcon("flyUse.jpg");
    Description[5] = "Use Case Diagram of FlyWeight Pattern";
    setDisplay(5);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Store FlyWeight Class Diagram"))
{
    Description[5] = "Class Diagram of FlyWeight Pattern";
    setDisplay(5);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Store FlyWeight Sequence Diagram"))
{
    Image[5] = new ImageIcon("flySeq.gif"); // reset image
    Description[5] = "Sequence Diagram of FlyWeight Pattern";
    setDisplay(5);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Store FlyWeight Object Interaction"))
{
    setDisplay(5);
    Image[5] = new ImageIcon("flyweight.jpg");
    Description[5] = "Object Interaction of FlyWeight Pattern";
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
else if (actionCommand.equals("Store FlyWeight DisCo Specification"))
{
    File sflyD = new File("c:\DPL\fly_disco.txt");
    save(sflyD);
}
else if (actionCommand.equals("Store Observer Use Case Diagram"))
{
    Image[1] = new ImageIcon("obsUse.jpg");
    Description[1] = "Use Case Diagram of Observer Pattern";
    setDisplay(1);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Store Observer Class Diagram"))
{
    Image[1] = new ImageIcon("obsC.gif");
    Description[1] = "Class Diagram of Observer Pattern";
    setDisplay(1);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Store Observer Sequence Diagram"))
{
    Image[1] = new ImageIcon("obsSeq.gif"); // reset image
    Description[1] = "Sequence Diagram of Observer Pattern";
    setDisplay(1);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Observer Object Interaction"))
{
    setDisplay(1);
Image[1] = new ImageIcon("Observer.jpg");
Description[1] = "Object Interaction of Observer Pattern";
getContentPane().add(descriptionPanel, BorderLayout.WEST);
setVisible(true);
}
else if (actionCommand.equals("Store Observer DisCo Specification"))
{
    File sobsD = new File("c:\DPL\obs_disco.txt");
save(sobsD);
}
else if (actionCommand.equals("Store Mediator Use Case Diagram"))
{
    Image[0] = new ImageIcon("medUse.jpg");
    Description[0] = "Use Case Diagram of Mediator Pattern";
    setDisplay(0);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Store Mediator Class Diagram"))
{
    Image[0] = new ImageIcon("medC.gif");
    Description[0] = "Class Diagram of Mediator Pattern";
    setDisplay(0);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Store Mediator Sequence Diagram"))
{
    Image[0] = new ImageIcon("medSeq.gif"); // reset image
    Description[0] = "Sequence Diagram of Mediator Pattern";
    setDisplay(0);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Mediator Object Interaction")) {
    Image[0] = new ImageIcon("mediator.jpg");
    Description[0] = "Object interaction of Mediator Pattern";
    setDisplay(0);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Store Mediator DisCo Specification")) {
    File smedD = new File("c:\DPL\med_disco.txt");
    save(smedD);
}
else if (actionCommand.equals("Store Meeting Use Case Diagram")) {
    Image[6] = new ImageIcon("meetUse.jpg");
    Description[6] = "Use Case Diagram of Meeting Pattern";
    setDisplay(6);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Store Meeting Class Diagram")) {
    Image[6] = new ImageIcon("meetC.gif");
    Description[6] = "Class Diagram of Meeting Pattern";
    setDisplay(6);
    getContentPane().add(descriptionPanel, BorderLayout.WEST);
    setVisible(true);
}
else if (actionCommand.equals("Store Meeting Sequence Diagram")) {
    Image[6] = new ImageIcon("medSeq.gif"); // reset image
Description[6] = "Sequence Diagram of Meeting Pattern";
setDisplay(6);
getContentPane().add(descriptionPanel, BorderLayout.WEST);
setVisible(true);
}
else if (actionCommand.equals("Meeting Object Interaction"))
{
Image[6] = new ImageIcon("meeting.jpg");
Description[6] = "Object interaction of Meeting Pattern";
setDisplay(6);
getContentPane().add(descriptionPanel, BorderLayout.WEST);
setVisible(true);
}
else if (actionCommand.equals("Store Meeting DisCo Specification"))
{
File smeetD = new File("c:\DPL\meet_disco.txt");
save(smeetD);
}
else if (actionCommand.equals("Store Managed Observer Use Case Diagram"))
{
Image[7] = new ImageIcon("mgdObsUse.jpg");
Description[7] = "Use Case Diagram of Managed Observer Pattern";
setDisplay(7);
getContentPane().add(descriptionPanel, BorderLayout.WEST);
setVisible(true);
}
else if (actionCommand.equals("Store Managed Observer Class Diagram"))
{
Image[7] = new ImageIcon("mgdObsC.gif");
Description[7] = "Class Diagram of Managed Observer Pattern";
setDisplay(7);
getContentPane().add(descriptionPanel, BorderLayout.WEST);
setVisible(true);
}
else if (actionCommand.equals("Store Managed Observer Sequence Diagram"))
{
  Image[7] = new ImageIcon("mgdObsSeq.gif"); // reset image
  Description[7] = "Sequence Diagram of Managed Observer Pattern";
  setDisplay(7);
  getContentPane().add(descriptionPanel, BorderLayout.WEST);
  setVisible(true);
}
else if (actionCommand.equals("Managed Observer Object Interaction"))
{
  Image[7] = new ImageIcon("mgdObs.jpg");
  Description[7] = "Object interaction of Managed Observer Pattern";
  setDisplay(7);
  getContentPane().add(descriptionPanel, BorderLayout.WEST);
  setVisible(true);
}
else if (actionCommand.equals("Store Managed Observer DisCo Specification"))
{
  File smgdD = new File("c:\DPL\mgdObs_disco.txt");
  save(smgdD);
}
else if (actionCommand.equals("Clear"))
{
  theText.setText(" ");
}
else if (actionCommand.equals("Exit"))
  System.exit(0);
if (obj == Open)
{
theText.setText(""); open();
}
if (obj == Save)
save();
if (obj == Clear)
theText.setText("");
if (obj == Cut) // cut
{
    theText.replaceRange("", theText.getSelectionStart(), theText.getSelectionEnd());
}
if (obj == Copy) // copy
{
    }
if (obj == Paste) // paste
{
    System.out.println("pasting");
    //paste();
    //Save.setEnabled(true);
}
/*
if (obj == Store)
{
    String queryText = "INSERT INTO Pattern "
    +"("PatternName,UseCase,Class,Sequence,DisCo) " +"VALUES('Strategy', 'Stra_useCase', 'stra_class', 'stra_seq', 'stra_disco')"
    ;
    query.setText(queryText);
}
if (obj == Retrieve)
{
    String queryText = " SELECT * " + "FROM Pattern" +" WHERE PatternName = 'Observer'";
    query.setText(queryText);
}
if (obj == Home)

{
    String queryText = "SELECT * " + "FROM Pattern";
    query.setText(queryText);
}
/*
if (obj == QS)
    System.exit(0);
/*
if (obj == Search)
    clickedSearch();
*/
if (obj == Input)
    {
        // modified = true;
        stringCount++;
        content[stringCount] = DataInput.getText().trim();  // update new data
    }
if (obj == StoreObj) // save image
    {
        saveImage/expImage, getFileName(FileDialog.SAVE));
    }
if (obj == RestoreObj) // RELOAD MODIFY AND SAVE IT BACK
    {
        SaveImgTest0 f = new SaveImgTest0();
        f.setVisible(true);
    }
if (obj == Preview)
    { /* create a LoadImageTest instance */
        LoadImageTest tr = new LoadImageTest();
        tr.setVisible(true);
    }
/*
if (obj == DisCo)
    getContentPane().add(new JScrollPane(theText),
    BorderLayout.CENTER);

102
/
if (obj == Quit)
System.exit(0);  
}
private void setBar()
  // set Bar
  Open = new ToolButton("Open", new ImageIcon("open.gif"));
  Save = new ToolButton("Save", new ImageIcon("disk.gif"));
  Clear = new ToolButton("Clear", new ImageIcon("erase.gif"));
  Quit = new ToolButton("Exit", new ImageIcon("stop.gif"));
  Copy = new ToolButton("Copy", new ImageIcon("copy.gif"));
  Cut = new ToolButton("Cut", new ImageIcon("cut.gif"));
  Paste = new ToolButton("Paste", new ImageIcon("paste.gif"));
  Home = new ToolButton("Home", new ImageIcon("home.gif"));
  //Search = new ToolButton("Query", new
  //Store = new ToolButton("Store", new ImageIcon("store.gif"));
  //Retrieve = new ToolButton("Retrieve", new
  //DisCo = new ToolButton("DisCo", new ImageIcon("d.gif"));
  StoreObj = new ToolButton("Save Image", new
  //Reload.gif")};
  RestoreObj = new ToolButton("Reload Image", new
  ImageIcon("Reload.gif")};
  Preview = new ToolButton("Preview", new
  ImageIcon("Reload.gif")};
  toolbar.add(Open);
  toolbar.add(Save);
  toolbar.add(Clear);
  Clear.setToolTipText("Clear the Content Panel");
  Save.setToolTipText("Save File");
  Quit.setToolTipText("Exit from program");
  Open.setToolTipText("Open a File");
  Copy.setToolTipText("Copy text");
  Cut.setToolTipText("Cut text");
  Paste.setToolTipText("Paste it");
Home.setToolTipText("Show All Patterns");
toolbar.add(Quit);
toolbar.add(Copy);
toolbar.add(Cut);
toolbar.add(Paste);
toolbar.add(StoreObj);
toolbar.add(RestoreObj);
toolbar.add(Preview);
Open.addActionListener(this);
Save.addActionListener(this);
Copy.addActionListener(this);
Cut.addActionListener(this);
Paste.addActionListener(this);
Home.addActionListener(this);
Clear.addActionListener(this);
Quit.addActionListener(this);
Preview.addActionListener(this);
StoreObj.addActionListener(this);
RestoreObj.addActionListener(this);
toolbar.addSeparator();
}

//-------------------------------------
public void loadList(java.awt.List list, String[] s) {
    list.removeAll();
    for (int i=0; i< s.length; i++)
        list.add(s[i]);
}

//-------------------------------------
public void addPanel(Panel ps, Component c) {
    Panel p = new Panel();
    ps.add(p);
    p.add(c);
}
private void clickedSearch() {
Results rs = db.Execute(query.getText());
String cnames[] = rs.getMetaData();
queryDialog q = new queryDialog(this, rs);
q.show();
}
*/
private void open()
{
    if (jFileChooser.showOpenDialog(this) ==
JFileChooser.APPROVE_OPTION)
    {
        open(jFileChooser.getSelectedFile());
    }
}

/**Open file with the specified File instance*/
private void open(File file)
{
    try
    {
        // Read from the specified file and store it in theText
        BufferedReader in = new BufferedReader(new
FileInputStream(file));
        byte[] b = new byte[in.available()];
in.read(b, 0, b.length);
theText.append(new String(b, 0, b.length));
in.close();

        // Display the status of the Open file operation
        l.setText(file.getName() + " Opened");
    }
catch (IOException ex)
```java
/** Error opening file. */

**Save file**
private void save()
{
    if (jFileChooser.showSaveDialog(this) == JFileChooser.APPROVE_OPTION)
    {
        save(jFileChooser.getSelectedFile());
    }
}

/** Save file with specified File instance */
private void save(File file)
{
    try
    {
        // Write the text in theText to the specified file
        FileOutputStream out = new FileOutputStream(file);
        byte[] b = (theText.getText()).getBytes();
        out.write(b, 0, b.length);
        out.close();

        // Display the status of the save file operation
        l.setText(file.getName() + " Saved ");
    }
    catch (IOException ex)
    {
        l.setText("Error saving " + file.getName());
    }
}
```
/*************** Copy()***************/
public void copy()
{
String s = theText.getSelectedText();
StringSelection contents = new StringSelection(s);
getToolkit().getSystemClipboard().setContents(contents, this);
}
public void paste()
{
boolean error = true;
Transferable t =
getToolkit().getSystemClipboard().getContents(this);
try {
if ((t != null) &&(t.isDataFlavorSupported(DataFlavor.stringFlavor)))
{
theText.replaceRange((String)t.getTransferData(DataFlavor.stringFlavor), theText.getSelectionStart(), theText.getSelectionEnd());
error = false;
} . .
} catch (UnsupportedFlavorException e)
{
}
}
catch (IOException e)
{
if (error)
{
theText.repaint();
theText.setText("Error: \	 Either the clipboard is empty");
}

} // paste()
public void textValueChanged(TextEvent evt) {
    //Save.setEnabled(true);
    GUI9.MODIFIED = true;
}

public void lostOwnership(Clipboard cp, Transferable contents) {
}

public void itemStateChanged(ItemEvent e) {
    setDisplay(c.getSelectedIndex());
}

public void setDisplay(int index) {
    // call setMethods() of descriptionPanel object
    descriptionPanel.setTitle(Title[index]);
    descriptionPanel.setImageIcon(Image[index]);
    descriptionPanel.setTextDescription(Description[index]);
}

/* to save object */
// Save the contents of the display under the specified file name
private void saveFile(String fileName) {
    if (fileName == null )
        return;
    try {
        FileOutputStream ostream = new FileOutputStream(new File(fileName));
        ObjectOutputStream p = new ObjectOutputStream (ostream);
        p.writeInt (numObjects);
        for (int i = 0; i < numObjects; i++) {
            p.writeObject(Obj[0]);
        }
    }
}
public void saveImage(BufferedImage buffi, String fileName) {
    if (fileName == null)
        return;
    try {
        /* write the jpeg to a file */
        FileOutputStream out = new FileOutputStream(new File(fileName));

        /* encodes expImage as a JPEG data stream */
        JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(out);
        JPEGEncodeParam param = encoder.getDefaultJPEGEncodeParam(buffi);
        param.setQuality(1.0f, false);
        encoder.setJPEGEncodeParam(param);
        encoder.encode(buffi);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

private void createImage() {
    System.out.println("Please wait a moment..." + nl);
    long start = System.currentTimeMillis();
    fc = new JFileChooser();
    int status = fc.showOpenDialog(this);
    fc.setCurrentDirectory(new File(System.getProperty("user.dir")));
    if (status == JFileChooser.APPROVE_OPTION) {
String file = fc.getSelectedFile().getPath();
try {
    image = Toolkit.getDefaultToolkit().getImage(file);
    mediaTracker = new MediaTracker(this);
    mediaTracker.addImage(image, 0);
    mediaTracker.waitForID(0);
    System.out.println("Image loading time:
    "+(System.currentTimeMillis() - start) + " ms" + nl);
} catch (InterruptedException e) {
    System.out.println("InterruptedException has occurred:" +
    e.getMessage());
    return;
}
width = image.getWidth(this);
height = image.getHeight(this);
System.out.println("Loading done!" + nl);
bufferImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
Graphics2D context = bufferImage.createGraphics();
context.drawImage(image, 0, 0, this);
System.out.println("ImageWidth: " + width + " ImageHeight: "+
    height + nl);
    jl.setText("Click on the picture...: ");
    image.flush();
}
*/
private String getFileName(int mode)
{
    String fileName;
    FileDialog dialog = new FileDialog( new Frame(), "Select file", mode);
    dialog.show();
    if (dialog.getDirectory() != null)
    {
        file = dialog.getDirectory() + System.getProperty("file.separator") + dialog.getFile();
        fileName = file;
        System.out.println("FileName: "+ fileName + nl);
        image = Toolkit.getDefaultToolkit().getImage(fileName);
        if (image != null)
        {
            width = image.getWidth(this);
            height = image.getHeight(this);
            System.out.println("Loading done!" + nl);
            bufferImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
            Graphics2D context = bufferImage.createGraphics();
            context.drawImage(image, 0, 0, this);
            System.out.println("ImageWidth: " + width + " ImageHeight: "+
                height + nl);
            jl.setText("Click on the picture...: ");
            image.flush();
        }
    }
}
fileName = dialog.getDirectory() + dialog.getFile();
return fileName;
} else {
    return null;
}

// inner class DrawingCanvas

class DrawingCanvas extends Canvas {

    final int RESET = 7;
    final int RECTANGLE = 6;
    final int LINE = 5;
    final int CLASS = 4;
    final int AGGREGATION = 3;
    final int INHERITANCE = 2;
    final int REMOVE = 1;
    final int TEXT = 0;
    int Last;
    int s = LINE;
    final int total = 100;
    int ObType[] = new int[total]; // object type
    Point begin[] = new Point[total];
    Point stop[] = new Point[total];
    Point angle;
    Point cur;
    Point p;
    int current = 0;

    DrawingCanvas() {
        setBackground(Color.white);
        final MouseAdapt myListener = new MouseAdapt();
        addMouseListener(myListener);
        final MouseMotionListener0 Motion = new
        MouseMotionListener0();
        addMouseMotionListener(Motion);
    }
void add()
{
begin[current]=angle;
stop[current]=cur;
ObType[current]= Last;
current++;
cur =null;
repaint();
}

public DrawableInterface drawClass(int x, int y, int w, int h, Graphics g)
{
g.drawRect(x, y, w, h);
g.drawLine(x, y+h/5, x+w, y+h/5);
g.drawLine(x, y+2*(h/3), x+w, y+2*(h/3));
return chosen ;
}

class Rect
{
	drawMethod(int x, int y, int w, int h, Graphics g)
{
	drawLine(x, y, x+n, y+n);
drawLine(x+n, y+n, x, y+2*n);
drawLine(x, y+2*n, x-n, y+n);
drawLine(x-n, y+n, x, y);
return chosen;
}

112
public DrawableInterface drawInheritance(int x, int y, int w, int h, Graphics g)
{
    int n = 20;
    g.drawLine(x, y, x-(n/2), y+n);
    g.drawLine(x-(n/2), y+n, x+(n/2), y+n);
    g.drawLine(x+(n/2), y+n, x, y);
    return chosen;
}
public DrawableInterface drawLine(int x1, int y1, int x2, int y2, Graphics g)
{
    g.drawLine(x1, y1, x2, y2);
    return chosen;
}
public DrawableInterface drawText(String s, int x1, int y1, Graphics g)
{
    g.drawString(s, x1, y1);
    return chosen;
}
public Dimension getPreferredSize()
{
    return new Dimension(width, height);
}
public void paint(Graphics g2)
{
    int IMAGE_WIDTH = 800;
    int IMAGE_HEIGHT = 600;
    Graphics2D gc = (Graphics2D)g2;
    expImage = new BufferedImage(IMAGE_WIDTH, IMAGE_HEIGHT, BufferedImage.TYPE_INT_RGB);

    /* draw into memory buffer */
    Graphics2D g = (Graphics2D) expImage.createGraphics();
    Rectangle r = getBounds();
if (Last == RESET)
{
    current = 0;
    g.setColor(Color.white);
}
else
{
    if (Last == REMOVE && current > 0)
    {
        current = current - 1;
        Last = -1;
    }
}
for (int i = 0, j = 0; i < current && j < current; i++, j++)
{
    s = ObType[i];
    switch (s)
    {
        case RECTANGLE:
            drawRectangle(begin[i].x, begin[i].y, stop[i].x, stop[i].y, g);
            case CLASS: Object[nums] =
                drawClass(begin[i].x, begin[i].y, stop[i].x, stop[i].y, g);
                break;
        case LINE: drawLine(begin[i].x, begin[i].y, stop[i].x, stop[i].y, g);
                break;
        case AGGREGATION: Object[nums] =
                drawAggregation(begin[i].x, begin[i].y, stop[i].x, stop[i].y, g);
                break;
        case INHERITANCE: Object[nums] =
                drawInheritance(begin[i].x, begin[i].y, stop[i].x, stop[i].y, g);
                break;
        case TEXT: g.setFont(new Font("Times New Roman", Font.PLAIN, 14));
                break;
    }
}
drawText(String.valueOf(content[j]), begin[i].x, begin[i].y, g);
    break;
  } // switch
} // for
if(cur != null)
{
  switch (Last)
  {
  case RECTANGLE: drawRectangle(angle.x, angle.y, cur.x, cur.y, g);
  case CLASS: drawClass(angle.x, angle.y, cur.x, cur.y, g);
    break;
  case AGGREGATION:
    drawAggregation(angle.x, angle.y, cur.x, cur.y, g);
    break;
  case INHERITANCE:
    drawInheritance(angle.x, angle.y, cur.x, cur.y, g);
    break;
  case LINE: g.drawLine(angle.x, angle.y, cur.x, cur.y);
    break;
  } // switch
} // if(cur != null)
gc.drawImage(expImage, 0, 0, this);
// paint()

public void redraw()
{
  angle.x = angle.y = cur.x = cur.y = 0;
  repaint();
}

class MouseAdapt0 extends MouseAdapter
{
  public void mousePressed(MouseEvent e)
  {
    angle = new Point(e.getPoint().x, e.getPoint().y);
p = new Point(e.getPoint().x, e.getPoint().y);
cur = new Point(e.getPoint().x, e.getPoint().y);
}

public void mouseReleased(MouseEvent e)
{
if (current < total)
add();
else
System.out.println("Error");
}
} // class MouseAdapt0

class MouseMotionListener0 extends MouseMotionAdapter
{
public void mouseDragged(MouseEvent e)
{
if (Last == LINE)
{
cur.x = e.getPoint().x;
cur.y = e.getPoint().y;
repaint();
return;
}
if (e.getPoint().y >= p.y)
{
angle.x = p.x;
cur.x = e.getPoint().x - angle.x;
}
else
{
angle.x = e.getPoint().x;
cur.x = p.x - angle.x;
}
if (e.getPoint().y >= p.y)
{
angle.y = p.y;
cur.y = e.getPoint().y - angle.y;
else
{
    angle.y = e.getPoint().y;
cur.y = p.y - angle.y;
}
repaint();
}
}//class MouseMotionListener
};// class DrawingCanvas

// inner class ToolBox
class ToolBox extends Panel
{
    DrawingCanvas C;
    JButton Line = new ToolButton("Line", new ImageIcon("Line.jpg"));
    JButton Rect = new ToolButton("Rectangle", new ImageIcon("rect.jpg"));
    JButton ClassRect = new ToolButton("Class", new ImageIcon("Class.jpg"));
    JButton Text = new ToolButton("Text", new ImageIcon("Text.jpg"));
    JButton Inheritance = new ToolButton("Inheritance", new ImageIcon("Gen.jpg"));
    JButton Aggregation = new ToolButton("Aggregation", new ImageIcon("Agg.jpg"));
    JButton Reset = new ToolButton("Reset All", new ImageIcon("erase.gif"));
    JButton REMOVE = new ToolButton("Undo", new ImageIcon("Undo.jpg"));

    public ToolBox(DrawingCanvas0 dc)
{this.C = dc;
setLayout(new GridLayout(11,0));
add(Line);
add(ClassRect);
add(Text);
add(Inheritence);
add(Aggregation);
add(REMOVE);
add(Reset);
Line.addActionListener(new java.awt.event.ActionListener(){
    public void actionPerformed(ActionEvent e) {
        Action(e, C.LINE);
    }
});
Rect.addActionListener(new java.awt.event.ActionListener(){
    public void actionPerformed(ActionEvent e) {
        Action(e, C.RECTANGLE);
    }
});
ClassRect.addActionListener(new java.awt.event.ActionListener(){
    public void actionPerformed(ActionEvent e) {
        Action(e, C.CLASS);
    }
});
Inheritence.addActionListener(new java.awt.event.ActionListener(){
    public void actionPerformed(ActionEvent e) {
        Action(e, C.INHERITANCE);
    }
});
}
Aggregation.addActionListener(new java.awt.event.ActionListener(){
    public void actionPerformed(ActionEvent e) {
        Action(e, C.AGGREGATION);
    }
});

REMOVE.addActionListener(new java.awt.event.ActionListener(){
    public void actionPerformed(ActionEvent e) {
        Action(e, C.REMOVE);
    }
});

Reset.addActionListener(new java.awt.event.ActionListener(){
    public void actionPerformed(ActionEvent e) {
        Action(e, C.RESET);
    }
});

Text.addActionListener(new java.awt.event.ActionListener(){
    public void actionPerformed(ActionEvent e) {
        Action(e, C.TEXT);
    }
});

void Action(ActionEvent e, int tag)
{
    int cnt=0;
    C.Last = tag;
    if(tag == C.REMOVE || tag == C.RESET)
        C.repaint();
    else
        C.redraw();
} // class Toolbox0
} // GUI0class
import java.awt.*;
import javax.swing.*;

public class DescriptionPanel0 extends JPanel {
    private JLabel Image = new JLabel();
    private JLabel Title = new JLabel();
    private JTextArea aText;

    public DescriptionPanel0() {
        JPanel panel = new JPanel();
        panel.setLayout(new BorderLayout());
        panel.add(Image, BorderLayout.CENTER);
        panel.add(Title, BorderLayout.SOUTH);

        // create a scrollPane object which is JTextArea
        JScrollPane scrollPane = new JScrollPane(aText = new JTextArea());
        Title.setHorizontalAlignment(JLabel.CENTER);
        Title.setFont(new Font("SansSerif", Font.BOLD, 16));
        aText.setFont(new Font("Serif", Font.PLAIN, 14));
        aText.setLineWrap(true);
        aText.setWrapStyleWord(true);
        scrollPane.setPreferredSize(new Dimension(200, 100));
        setLayout(new BorderLayout());
        add(panel, BorderLayout.EAST);
    }

    public void setTitle(String title) {
        Title.setText(title);
    }

    public void setImageIcon(ImageIcon icon) {
        Image.setIcon(icon);
    }

    public void setTextDescription(String text)
{  
aText.setText(text);
}
}  // DescriptionPanel class
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
import java.io.*;
import java.awt.Canvas;
import java.awt.Canvas.*;

public abstract interface DrawableInterface {
    public abstract void addToCanvas(DrawingCanvasP dp);
    public abstract void draw(java.awt.Graphics g);
} // DrawableInterface
import java.awt.*;
import java.awt.event.*;
import java.util.*;
//swing classes
import javax.swing.text.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;

public class ToolButton extends JButton {
    public ToolButton(String caption, Icon img) {
        super(caption, img);
        setMargin(new Insets(0,0,0,0));
        setSize(25,25);
    }
}
import java.awt.*;
import java.awt.event.*;
public class WindowCloser0 extends WindowAdapter
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
} // class WindowCloser0 extends
APPENDIX B

HTML CODE
Welcome to the On Line Design Patterns Library

send your comments or suggestions to Philip

<a href="mailto: weiwu7@hotmail.com"> weiwu7@hotmail.com </a>
<HTML><HEAD><TITLE> An On Line Design Patterns Library </TITLE></HEAD>
<BODY GCOLOR="white" leftmargin="0" topmargin="0" TEXT="black" LINK="#003366" VLINK="navy" ALINK="#003366">
<font face="Arial,'Courier New', Tahoma">
<table border="1" cellpadding="0" cellspacing="0" height="100%" width="155">
<tr><td valign="top" height="62" width="155">
<table bgcolor="Black" border="0" cellpadding="0" cellspacing="0" height=62 width="100%">
<tr><td><font color="white" face="Arial,'Courier New', Tahoma" size="2">
<div align="center"> On Line Design Patterns Library </div></font></td></tr>
</table></td></tr></table>
<br><br><font size="-1">&nbsp;</font><h3>Generic Design Patterns</h3><br>
<strong>Creational Design Patterns</strong><br><br>
<a href="factory.html" target="menu" title="Factory Patterns">Factory Patterns</a><br>
<br>
<strong>Structural Design Patterns</strong><br><br>
<a href="bridge.html" target="menu" title="Bridge Patterns">Bridge Patterns</a><br>
<br>
<a href="composite.html" target="menu" title="Composite Patterns">Composite Patterns</a><br>
<br>
<a href="facade.html" target="menu" title="Facade Patterns">Facade Patterns</a>
</font></BODY></HTML>
Behavioral Design Patterns

Mediator Patterns

Observer Patterns

Strategy Patterns

Domain Specific Design Patterns

Master-Slave Patterns

Combination of patterns

Managed Observer Pattern

CSUSB Computer Science Department

About Philip Wu

An On Line Design Patterns Library

STYLE TYPE="text/css">
<!--A:link {color:"#003399"}
On Line Design Patterns

Library

Generic Design Patterns

Creational

Structural

Behavioral
On Line Design Patterns Library

Generic Design Patterns

Creational Design Patterns

Structural Design Patterns

Behavioral Design Patterns
Objects interaction

Class Diagram

DISCO Specification

Store

Retrieve

Observer Patterns

Strategy Patterns

Domain Specific Design Patterns

Master-Slave Patterns

Combination of patterns

Managed Observer Pattern

Objects interaction

Class Diagram

DISCO Specification

Store

Retrieve

Observer Patterns

Strategy Patterns

Domain Specific Design Patterns

Master-Slave Patterns

Combination of patterns

Managed Observer Pattern

Objects interaction
<table border=0 align="left" valign="top"><tr><td width="1"></td></tr><table border=1 bordercolor="navy" align="left" valign="top"><tr><td><table border=0 align="left" valign="top"> <tr align="left" valign="top"> <td width="1"></td></tr><li><a href="http://www.csci.csusb.edu" target="main" title="CSUSB Computer Science Department"><font size="-1">CSUSB Computer Science Department</font></a></li><a href="philip.html"><font size="-1">About Philip Wu</font></a></td></tr></table></table></table>
An On Line Design Patterns Library

Generic Design Patterns

Creational Design Patterns

Structural Design Patterns

Behavioral Design Patterns

Composite Patterns

Objects interaction
REFERENCES


