Riddhi Damani
Student ID: 2007710
Master's in Computer Science

DePaul University
Instructor: Professor Jeffrey Sharpe
SE 450: Object Oriented Software Development

Report
JPaint Project in Java

- **List of missing features, bugs, and miscellaneous notes**.

Implemented and Missing Features:

Sr. No:	Functionality (Requirement)	Successfully Implemented?	Missing Features
1	Draw Rectangle, Ellipse, and Triangle	Successfully Implemented	No
2	Draw Shapes with various colors	Successfully Implemented	No
3	Draw Shapes with various outline – a) Outline Only – Only shape outline will be drawn. Use Primary Color to draw this. b) Filled-In – Only the inside of the shape will be drawn – there will be no visible outline. Use Primary Color to draw this. c) Outline and Filled-In – Both the inside and the outline will be drawn. Use Primary Color for the inside and Secondary Color for the outline.	Successfully Implemented	No
4	Select a Shape (using collision detection algorithm)	Successfully Implemented	No
5	Move a Shape	Successfully Implemented	No
6	Copy	Successfully Implemented	No
7	Paste	Successfully Implemented	No
8	Delete	Successfully Implemented	No
9	Outline Selected Shapes	Successfully Implemented	No
10	Undo Operation Draw, Move, Paste, Delete, Group, Ungroup	Successfully Implemented	No
11	Redo Operation Draw, Move, Paste, Delete, Group, Ungroup	Successfully Implemented	No
12	Group Draw, Select (wt Outline), Move, Copy, Paste, Delete, Undo, Redo, Group again	Partial	Partial
13	UnGroup Draw, Select (wt Outline) , Undo, Redo, Group again	Successfully Implemented with a minor bug	Outline is not correctly drawn as mentioned below.

Riddhi Damani
Student ID: 2007710
Master's in Computer Science

DePaul University
Instructor: Professor Jeffrey Sharpe
SE 450: Object Oriented Software Development

Bugs:

1) When I run my program and if I resize or minimize the paintCanvas screen, and then open back, all the shapes that I have drawn on the canvas disappears i.e. canvas gets cleared. However, if I click on the screen, all the drawn shapes appear back strangely*. (Bug existed prior to project commencement)*
2) While selecting the shapes and pressing on 'Group' button, the group gets selected perfectly. But, the problem lies in the drawing of the group outline. Sometimes, the outline is calculated properly and is drawn over the object, but most of the times there is a discrepancy in the **group select outline**.
3) While moving a grouped object, the object moves perfectly with the calculated offset; but the **outline** stays on the screen and does not move along with the shapes. I have been working on this for quite long; but it seems like I have to change my design in all in order to incorporate proper outline over the group move functionality.
4) Group copy works perfectly fine; but **group paste** throws me a NullPointer exception as I have my strategy pattern in place for my leaf nodes. I have been working on this for quite some time as well; but with no luck with resolution.

Miscellaneous notes:

Group and UnGroup Operation Notes: Though few bugs persist in the group functionality. Below operations are working correctly:

1) If you try to select two or more objects and click on group – Group is drawn perfectly.
2) Group of group is also selected and drawn perfectly alright.
3) Group + individual shapes is selected and perfectly drawn as well.
4) Moving the selected group works fine; the group moves as per the calculated xDelta and yDelta position. But, problem lies with the outline as mentioned above.
5) Ungrouping the above group scenarios – work perfectly fine.
6) Copy of the Group works fine.
7) Deleting the Group works fine too.
8) Group Undo and Redo works fine too.
9) Ungroup Undo and Redo works fine too.
10) If you are ungrouping the grouped object, the outline on the ungroup objects gets drawn only if u select it again.

Riddhi Damani
Student ID: 2007710
Master's in Computer Science

DePaul University
Instructor: Professor Jeffrey Sharpe
SE 450: Object Oriented Software Development

- **Notes on design and UML Diagrams all of the patterns implemented (5 Design Patterns)**

Five Software Design Patterns that I have managed to implement as a part of JPaint application are as below:

1) **Singleton Design Pattern**

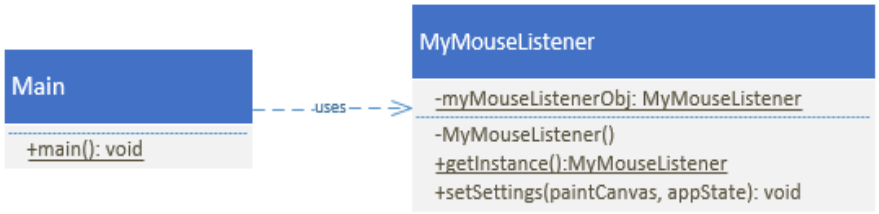
Classes Involved: Main.java, MyMouseListener.java

Explanation: MyMouseListener class has a private constructor and have a static instance of itself. This class provides a static method named *getInstance ()* to get its static instance to my outside class i.e. Main.java. Main.java will use this method to receive the *myMouseListenerObj* and thus, invoke the *setSettings (paintCanvas, appState)* method.

Why this pattern? : As there was requirement to use only a single instance of the mouse listener class, I decided go with Singleton Design Pattern.

What problem it solved? : It helped me solve two problems: a) myMouseListener class has just a single instance (myMouseListenerObj) b) It provided me a global access point to that instance.

UML Class Diagram:



2) **Strategy Design Pattern**

Classes Involved: DrawShape.java, DrawRectShape.java, DrawEllipseShape.java, DrawTriangle.java, DrawCommand.java

Interface Involved: IShape.java

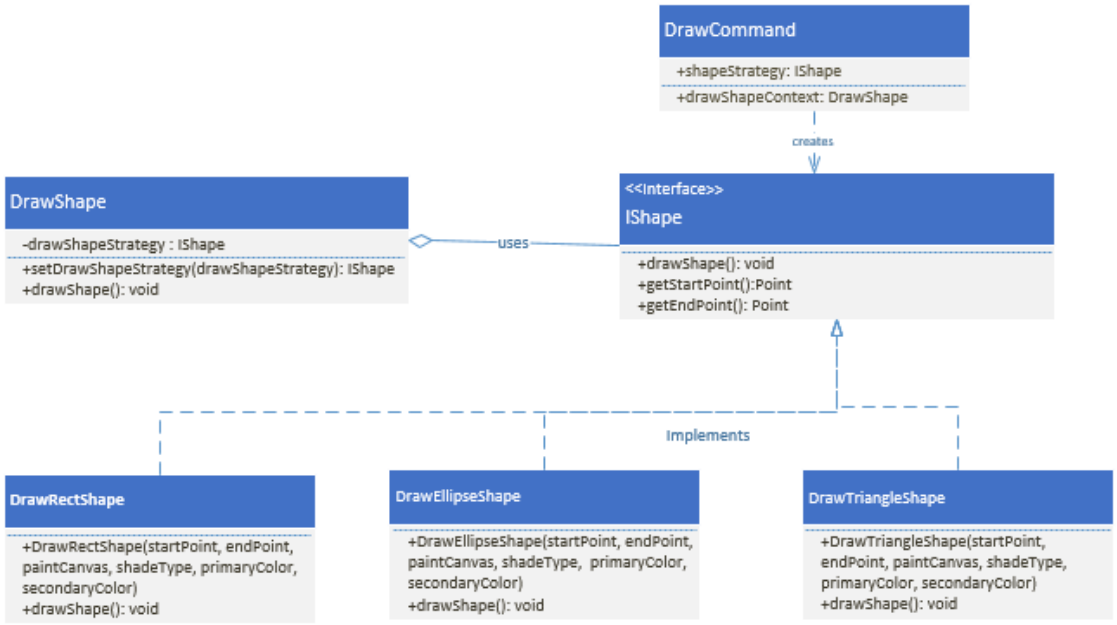
Explanation: IShape interface acts as a strategy interface here which defines the *drawShape ()* action. This interface is then implemented by the three concrete classes – DrawRectShape, DrawEllipseShape and DrawTriangleShape. DrawShape class sets the context of the strategy. The DrawCommand class will use the context and the strategy objects to determine change in context behavior based on strategy it deploys or uses.

Why this pattern? : All the shapes had almost similar data i.e. properties required to build them, for instance: startPoint, endPoint, paintCanvas, shadeType, primaryColor and secondaryColor; the only difference was in their behaviors. So, strategy pattern was a perfect fit for its implementation.

What problem it solved? : It helped me separate the behaviors of each shapes into different classes and strategies; and thus allowing me to achieve easy maintainability of my code. Using this pattern will allow

the user to add any other shape's strategies in future with much ease. Also, it helped me to achieve the Single Responsibility Principle where each class performs only one kind of operation.

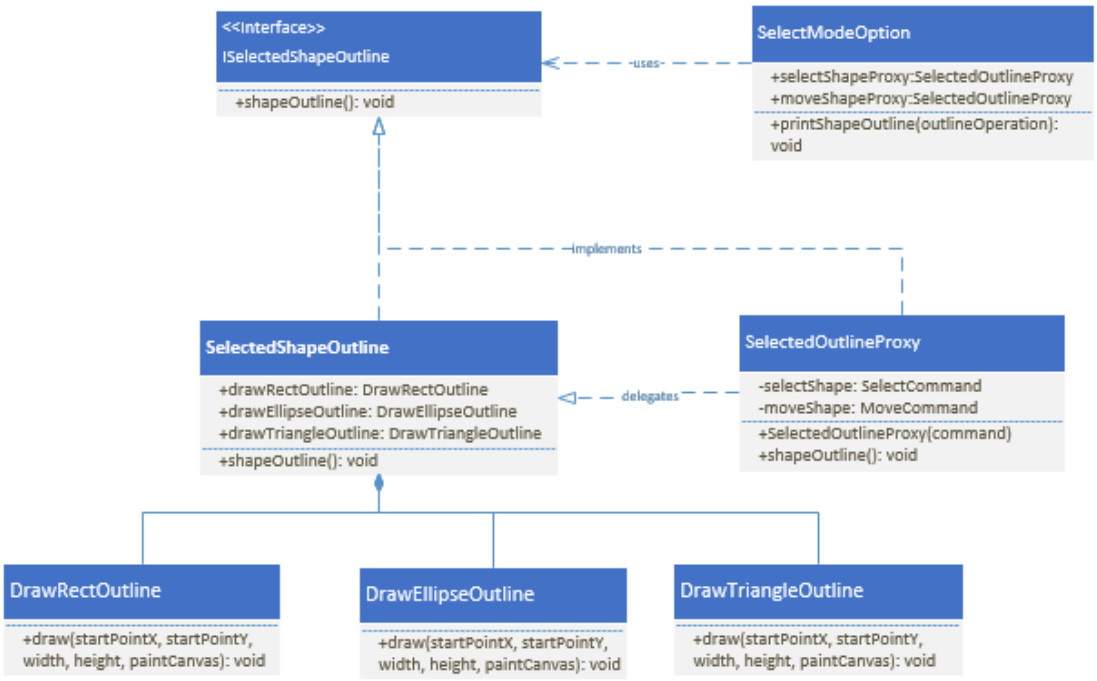
UML Class Diagram:



3) Proxy Design Pattern

Classes Involved: SelectModeOption.java, SelectedOutlineProxy.java, SelectedShapeOutline.java, DrawRectOutline.java, DrawEllipseOutline.java, DrawTriangleOutline.java
Interface Involved: ISelectedShapeOutline.java
Explanation: ISelectedShapeOutline interface defines the *shapeOutline ()* method; which is implemented by the concrete classes SelectedOutlineProxy.java and SelectedShapeOutline.java. The proxy class here prevents the outline to be drawn if the shapes are not selected. It acts as a protection layer. SelectModeOption.java uses the proxy class object to invoke the *printShapeOutline (ISelectedShapeOutline outlineOperation)* method, who is unaware of the concrete type of the shapeOutline object. This method only knows that it has received an ISelectedShapeOutline object. The other 3 concrete classes - DrawRectOutline.java, DrawEllipseOutline.java, DrawTriangleOutline.java performs the actual logic of drawing the outline on the canvas. I used it to provide an encapsulation layer.
Why this pattern? : To add an additional 'outline' functionality to my selected shapes. This pattern helped me enforce a condition of whether a shape is selected or not before actually executing the draw outline. Also, this pattern promotes the separation of responsibilities.
What problem it solved? : Due to the implementation of the proxy class, I was able to put a check on the variables that helped me identify whether the shape is selected or moved before actually drawing the shape's outline on the paintCanvas.

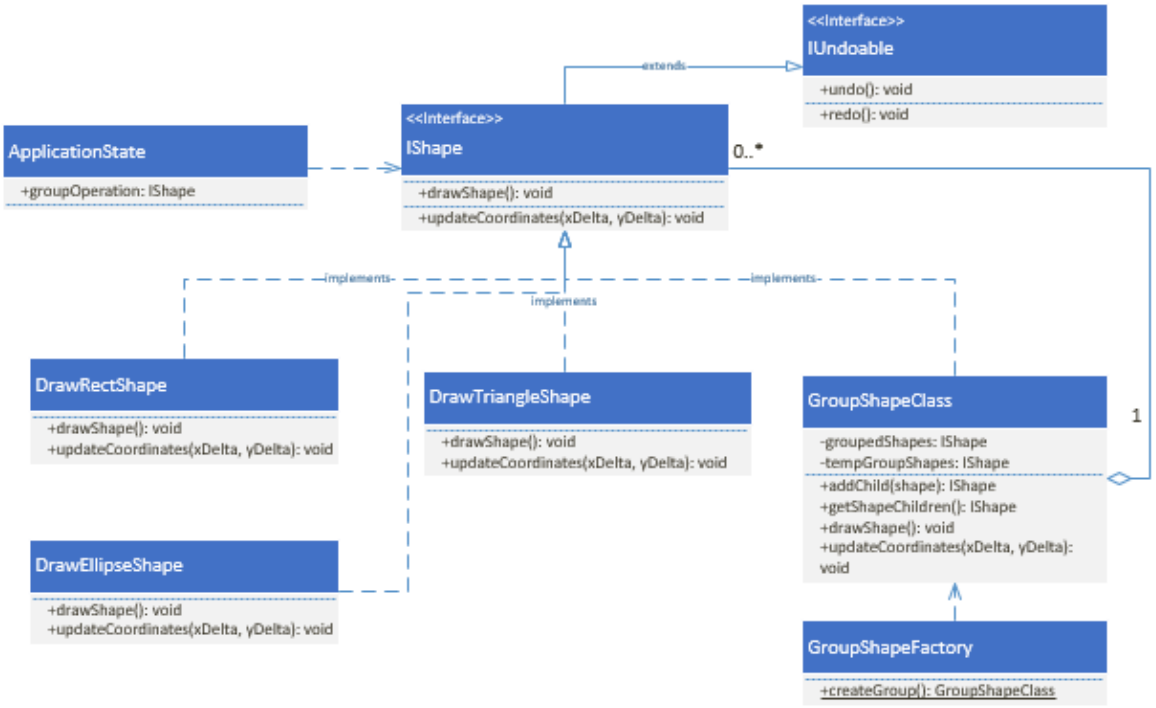
UML Class Diagram:



4) Composite Design Pattern

Classes Involved: ApplicationState.java, GroupShapeFactory.java, GroupShapeClass.java, DrawRectShape.java, DrawTriangleShape.java, DrawEllipseShape.java
Interface Involved: IShape.java, IUndoable.java
Explanation: ApplicationState class is responsible for invoking the createGroup () method defined in the GroupShapeFactory class. This class creates a group object of the selected shapes on the canvas. The selected individual shapes are then removed from the master shape list and the group object is added to it. Thus, treating the group object as an individual object. The IShape and IUndoable method is implemented in all the leaf nodes as well as the composite class.
Why this pattern? : Since I was required to represent objects in form of a tree structure and wanted the group shape object to be treated in similar way as the single object of each shape i.e. Rectangle, Triangle and Ellipse; Composite Design pattern suited best in this scenario.
What problem it solved? : This pattern helped me to avoid code smell in my project, by letting me implement the common interface i.e. IShape between leaf node and the composite node. It made easier to add new kinds of components and provided flexibility of structure with manageable class and interfaces.

UML Class Diagram:



5) Null Object Design Pattern

Classes Involved: DrawCommand.java, ExecuteNullShape.java
Interface Involved: IShape.java
Explanation: ExecuteNullShape class implements all the IShape methods, just like the other three concrete classes – DrawRectShape, DrawEllipseShape and DrawTriangleShape that are created by the DrawCommand class. This class basically performs default action of doing nothing.
Why this pattern? : In order to avoid conditional complexity by using null objects rather than primitive null checks during Draw Command. I used this pattern to provide a default behavior of doing nothing.
What problem it solved? : This pattern solved my problem of handling NullReferenceExceptions when methods are called in my Draw Command.

UML Class Diagram:

<u>Additional Design Patterns Implemented:</u>

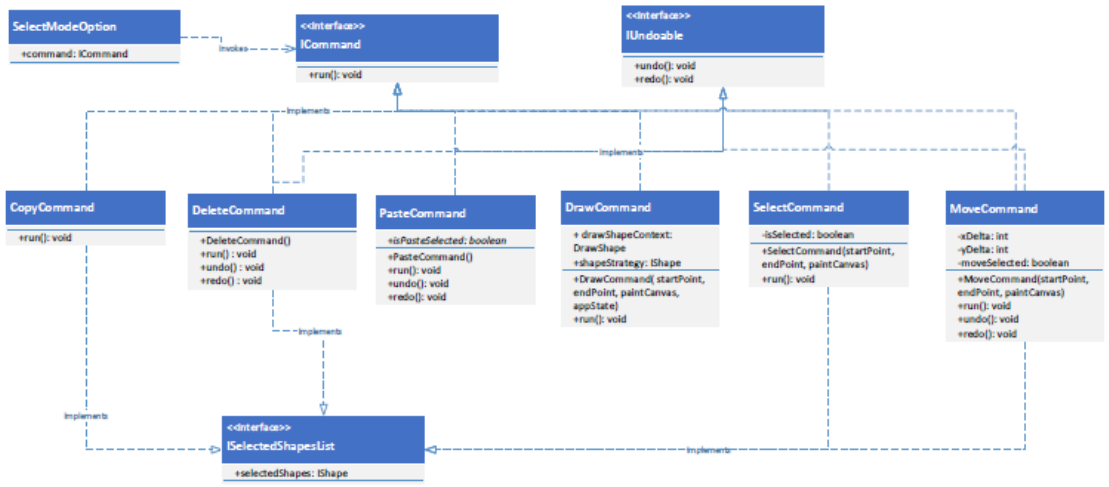
6) Command Design Pattern

Classes Involved: MyMouseListener.java**,** SelectModeOption.java**,** DrawCommand.java**,** SelectCommand.java**,** MoveCommand.java
Interface Involved: ICommand.java, IUndoable.java
Explanation: ICommand interface defines a method run () which is implemented by six concrete classes – CopyCommand, DeleteCommand, PasteCommand, SelectCommand, MoveCommand, and DrawCommand. The class SelectModeOption.java acts as an invoker object and takes in and invokes the command.
Why this pattern? : Since this pattern is a data driven pattern and allowed me to wrap a request under an object as a command and thus, executes the respective command; it was a perfect fit for my scenario here.
What problem it solved? : It helped me separate the object that invoked the operation from the object that actually performed the operation. Also, it helped me make my class easier to add new commands without hampering the existing classes. Thus, providing code flexibility.

Riddhi Damani
Student ID: 2007710
Master's in Computer Science

DePaul University
Instructor: Professor Jeffrey Sharpe
SE 450: Object Oriented Software Development

UML Class Diagram:



➢ Successes and Failures

Being my first quarter in DePaul and having around 4 years of professional work experience in mainframe technologies, I was unaware about the prominence of software design patterns and SOLID principles.

SE 450 under the guidance of Prof. Jeffrey Sharpe have been a great learning experience for me – not only I gained immense knowledge on appropriate usage of object-oriented practices, but I have also enhanced my skills on Core Java and Java SWING.

Implementing the 5 software design patterns in agile approach - Singleton pattern, Strategy Pattern, Proxy Pattern, Command Pattern and Null object pattern is what I consider a huge success for my project. Also, my potential to utilize SOLID principles developed extensively throughout the sprints. Understanding the core of concrete relationships between classes like aggregation, composition, dependency, interface implementation has assuredly been a good takeaway. I consider this as a part of my project successes.

Successes:

- ✓ While commencing JPaint application in the first sprint, I faced tuff time drawing a rectangle using a command pattern. But, listening to the professor's lecture over and over again along with the project suggestions that he made, helped me articulate my thoughts and logic in a structured manner.

- ✓ During Sprint 2, I faced difficulty in deciding the pattern for drawing the shapes on the canvas – factory or the strategy design pattern. After going through professor's suggestion multiple times and also

understanding the fact that the shapes had similar properties, only behavior were different in each of them, I choose to use Strategy design pattern. Also, figuring out the collision detection algorithm, how actually it works was quite a task for me. It actually took me 2-3 days to understand the logic behind it and thus, successfully implementing it on my select command.

- ✓ Sprint 3 was pretty smooth for me as it involved the implementation of Copy, Paste and Delete and Outline. Outline was a little struggle for me here as initially I decided to use decorator pattern for it, but I was failing to wrap the outline object with the shapes object. Thus, changed my decision to use the proxy design pattern where I decided to place the flag (Boolean values) as condition for allowing to draw an outline over a shape. This went pretty smooth.

Failures:

- ✓ What went wrong with my project was the implementation of the Group operation in Sprint 4. It took me a week to decipher Group functionality using composite pattern – after repeatedly watching the Composite pattern lecture by the professor and immense research on composite pattern implementation for composite object, I finally was able to crack the Group draw shape, group move, group undo, group redo and ungrouping. But, still I feel disappointed to not crack the group select outline and paste properly. Although the outline draws alright sometimes, but most of the time it fails to capture the correct width and height.

Regardless of minor bugs in my JPaint application, I feel prodigious about what I have accomplished and learnt so far. I completely abide to the fact that there are wide areas in my application that can be refactored and encapsulated for being eligible as a good software design solution. I will certainly try my best to work on it in later versions as a part of my personal practice.

In a nutshell, this course has been given me a solid kickstart to pursue my career in full-stack application development by teaching me to build design specific solutions rather than just providing a software solution.