# The IPaint Application does the following:

☐ Draws Rectangle, Ellipse, and Triangle shapes on the canvas

☐ Draws the shapes with various colors

☐ Draws the shapes with shading types of either, filled in, filled in and outline, or outline

Instructions:

|  |  |
| --- | --- |
| Select a shape | Single select  In select mode clicking on a single shape on the canvas will cause that shape to be selected.  Multi-select  In Select mouse mode, select any shapes that are touched while clicking and dragging will be selected |
| Move a shape | A shape or a group of shapes, has to be selected before it can be moved  In Move Mouse Mode, clicking and dragging will offset any Selected shapes by the amount of mouse moves  Move behaves the same whether a single shape or a group of shapes are selected |
| Copy and paste a shape | Copy and paste works together. A shape must be copied to the clipboard before it can be pasted on the canvas |
| Group and Ungroup shapes | A shape or shapes must be selected before it can be added to a group. Shapes grouped together are operated on as if they are one shape. Copy, paste, move, undo, and redo works on grouped shapes. Groups can be nested. |

# Technical Jargon:

Collision detection is performed to identify which shapes are selected and which shapes should be grouped or ungrouped. The detection is done by generating a bounding box around the shapes.

## Extra Credit

|  |
| --- |
|  |
| When shape(s) are selected a temporary box will flicker on the screen around the shapes, to indicate the size and location of the bounding box surrounding the shape(s). |
| When shape(s) are grouped two boxes will appear on the screen. One box indicates the area of the canvas where the shapes were selected from. The other box indicates the area occupied by the grouped shapes. Each bounding box is also labeled.    Since this is not a real world app, I kept the system.out.println() log messages in the app. If I had more time I would have created a proper log function and some testing units. |

## Design patterns used in Application

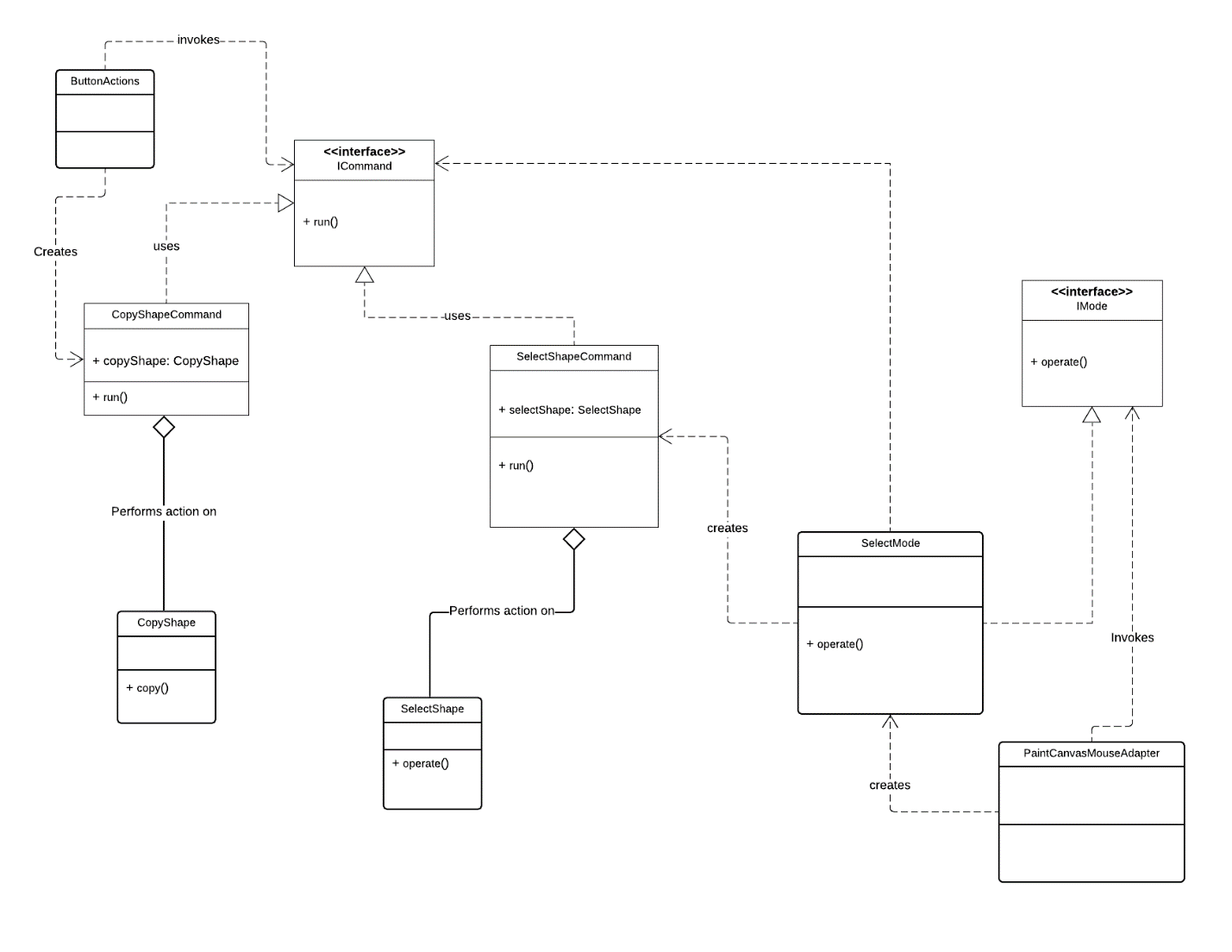
### Command Design Pattern

|  |
| --- |
|  |
| The command pattern encapsulates in an object all the data required for performing a given action (command), including what method to call, the method’s arguments, and the object to which the method belongs |
| The command pattern is used in the application to store all the information required for executing the actions on the canvas. Undo and Redo is also implemented as a part of the command pattern |
|  |

#### The Application performs the following commands

|  |  |
| --- | --- |
| Command | Undoable (Yes/No) |
|  |  |
| Select | No:  Clicking outside a shape will unselect all selected shapes |
| Copy | No:  The next click on the copy button will replace the previously copied shapes on the clipboard |
|  |  |
| Undo | No:  A click on the Redo button will revert the changes made by the undo button |
| Redo | No:  A click on the Undo button will revert the changes made by the redo button |
|  |  |
| ☐ Draw | Yes |
| ☐ Move | Yes |
| ☐ Paste | Yes |
| ☐ Delete | Yes |
| ☐ Group | Yes |
| ☐ Ungroup | Yes |

|  |
| --- |
|  |

****

### Patterns using other patterns

In this project the strategy, null object, and static factory were used together to create a shape on the canvas

#### Strategy Design Pattern

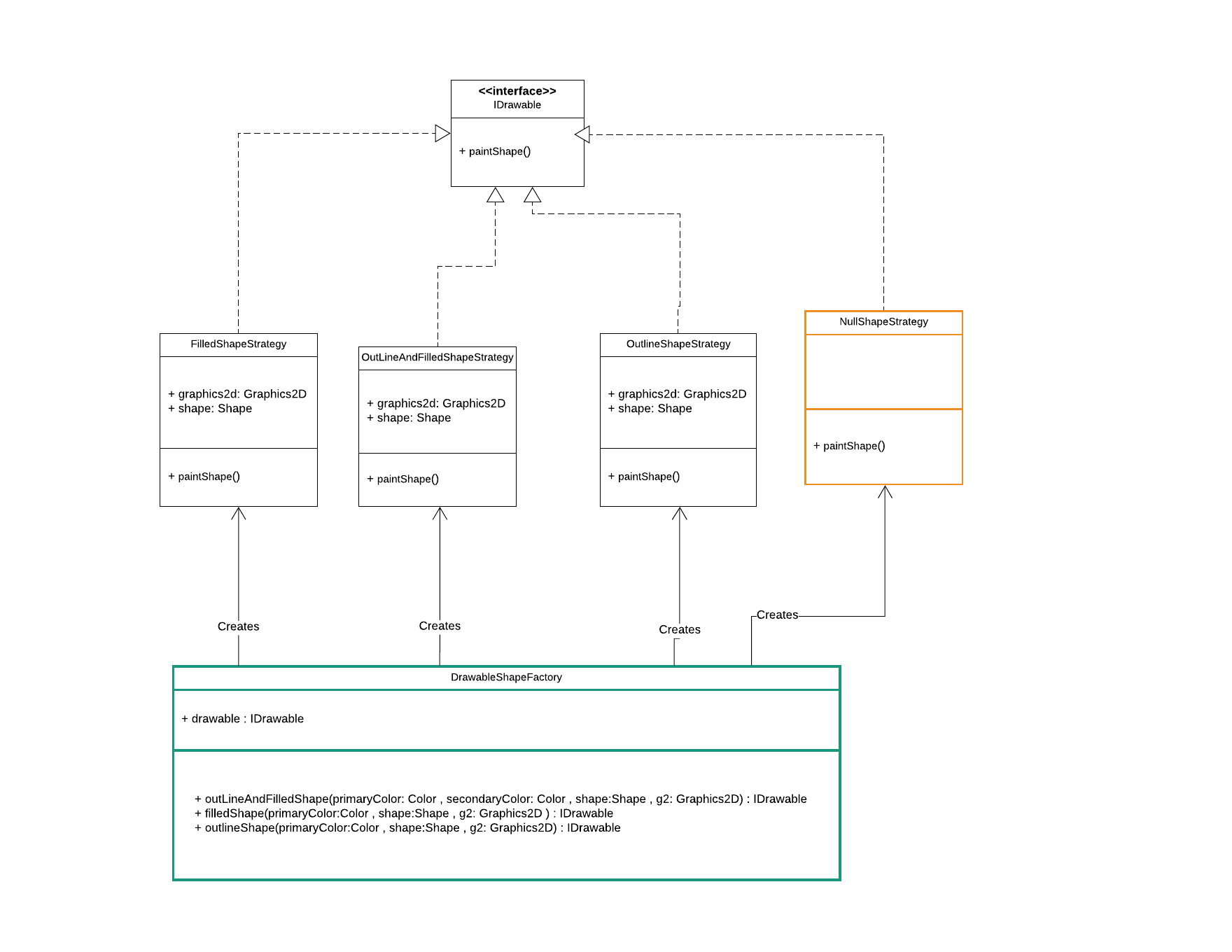
|  |
| --- |
| The Strategy Design Pattern defines a family of algorithms, encapsulate each one, and make them interchangeable |
| This pattern is used in conjunction with the DrawableShapeFactory and ShapeTypeFactory static factories. A new algorithm is selected at runtime. |
|  |

#### Null Object Design Pattern

|  |
| --- |
| The Null object pattern is a design pattern that simplifies the use of dependencies that can be undefined |
| A NullShapeStrategy null object was created as one of the strategies belonging to the IDrawable algorithm |
|  |

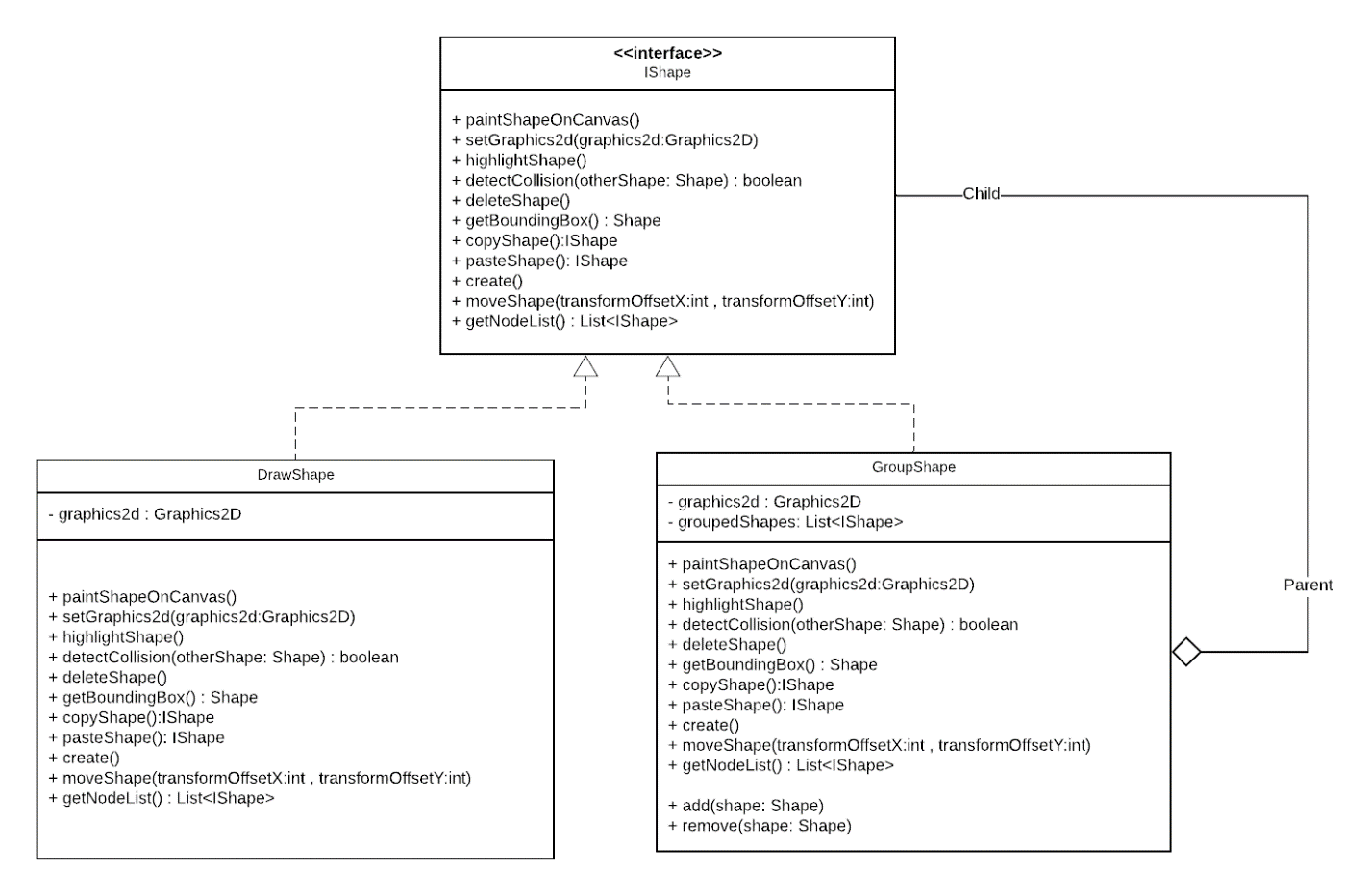
#### Static Factory Design Pattern

|  |
| --- |
| A Static factory a static method that returns an instance of a class. |
| There are two static factories in this application, DrawableShapeFactory and ShapeTypeFactory   * There is a DrawableShapeFactory which creates the outline, filled-in, outline and filled in shading types for the shapes. * There is a ShapeTypeFactory which creates the different types of shapes that can be drawn on the canvas. * These static factories utilize algorithms created with the strategy pattern. DrawableShapeFactory is responsible for creating a new instance of the IDrawable algorithm at runtime.ShapeTypeFactory is responsible for creating a new instance of the IShapeTypeStrategy algorithm at runtime and produce a shape.   Using a factory makes is easy to change the names of the instantiated classes, since the client code does not have to know how to create a concrete class. |
|  |

****

### Composite Design Pattern

|  |
| --- |
|  |
|  |
| The composite pattern was used in the application because the commands needed to work the same on groups.  This pattern solved the problem where the difference between individual shapes and shapes that are grouped together needed to be ignored. |

****

### Observer Design Pattern

|  |
| --- |
| The Observer defines a one-to-many relationship so that when one object changes state, the others are notified and updated |
|  |
| The observer pattern is used in this project to notify the paintCanvas when changes were made to the shapes. The paintCanvas then redraws the shapes on the canvas |

### Singleton Design Pattern

|  |
| --- |
|  |
| The singleton pattern is a software design pattern that restricts the instantiation of a class to one "single" instance. This is useful when exactly one object is needed to coordinate actions across the system |
| The singleton design pattern is used to create the object used to store the currently selected shapes on the canvas. I choose to use this pattern instead of a static class so that I could use an interface, as well as have access to methods and properties without having to declare all of them static. The SelectBoundingBox class uses this pattern. |
|  |

# Issues with this project.

Most of the issues I experienced with this project involved creating the UML diagram. The definition for the diagrams is often confusing and to be honest I am not 100% sure that I used the correct signs. I made a note for myself about what the signs mean, but I am not sure it is accurate.

