**Paint Report**

**Introduction:**

**Microsoft Paint**  is a simple [computer graphics](https://en.wikipedia.org/wiki/Computer_graphics) program that has been included with all versions of [Microsoft Windows](https://en.wikipedia.org/wiki/Microsoft_Windows). The program mainly opens and saves files as [Windows bitmap](https://en.wikipedia.org/wiki/Windows_bitmap) (24-bit, 256 color, 16 color, and [monochrome](https://en.wikipedia.org/wiki/Monochrome), all with the .bmp extension), [JPEG](https://en.wikipedia.org/wiki/JPEG), [GIF](https://en.wikipedia.org/wiki/Graphics_Interchange_Format) (without animation or transparency, although the [Windows 98](https://en.wikipedia.org/wiki/Windows_98) version, a [Windows 95](https://en.wikipedia.org/wiki/Windows_95) upgrade, and the [Windows NT4](https://en.wikipedia.org/wiki/Windows_NT4) version did support the latter), [PNG](https://en.wikipedia.org/wiki/Portable_Network_Graphics) (without [alpha channel](https://en.wikipedia.org/wiki/Alpha_compositing)), and single-page [TIFF](https://en.wikipedia.org/wiki/TIFF). The program can be in color mode or two-color [black-and-white](https://en.wikipedia.org/wiki/Black-and-white), but there is no [grayscale](https://en.wikipedia.org/wiki/Grayscale) mode. For its simplicity, it rapidly became one of the most used applications in the early versions of Windows—introducing many to painting on a computer for the first time—and is still widely used for very simple image manipulation tasks.

-----------------------------------------------------------------------------------------------

The aim of this project is to design a simple paint application using java programming language by implementing an OOP Design with the aid of *design patterns*.

**Design patterns:**

In [software engineering](https://en.wikipedia.org/wiki/Software_engineering), a software design pattern is a general reusable solution to a commonly occurring problem within a given context in [software design](https://en.wikipedia.org/wiki/Software_design). It is not a finished design that can be transformed directly into [source](https://en.wikipedia.org/wiki/Source_code) or [machine](https://en.wikipedia.org/wiki/Machine_code) code. It is a description or template for how to solve a problem that can be used in many different situations. Design patterns are formalized [best practices](https://en.wikipedia.org/wiki/Best_practice) that the programmer can use to solve common problems when designing an application or system.

[Object-oriented](https://en.wikipedia.org/wiki/Object-oriented) design patterns typically show relationships and [interactions](https://en.wikipedia.org/wiki/Interaction) between [classes](https://en.wikipedia.org/wiki/Class_(computer_science)) or [objects](https://en.wikipedia.org/wiki/Object_(computer_science)), without specifying the final application classes or objects that are involved. Patterns that imply mutable state may be unsuited for [functional programming](https://en.wikipedia.org/wiki/Functional_programming) languages, some patterns can be rendered unnecessary in languages that have built-in support for solving the problem they are trying to solve, and object-oriented patterns are not necessarily suitable for non-object-oriented languages.

Design patterns may be viewed as a structured approach to [computer programming](https://en.wikipedia.org/wiki/Computer_programming) intermediate between the levels of a [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm) and a concrete [algorithm](https://en.wikipedia.org/wiki/Algorithm).

**Project Discussion**

**Design patterns used:**

1-Factory design pattern:

Factory pattern is one of most used design pattern in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

Implementation*:* We’ve created a shape interface and concrete classes implementing it which are : rectangle , square , circle , ellipse , triangle , rightTriangle , line.

A ShapeFactory class has static method getShape to create a new shape and getCopyShape to get a copy of a shape.

2-Decorator design pattern:

Decorator pattern allows a user to add new functionality to an existing object without altering its structure. This type of design pattern comes under structural pattern as this pattern acts as a wrapper to existing class.

This pattern creates a decorator class which wraps the original class and provides additional functionality keeping class methods signature intact.

Implementation*:* We’ve created a ShapeDecorator abstract class and 2 concrete classes , FillColor and BorderColor, that extend it and they take a shape object to add fill color , border color and set stroke width respectively.

3-Object pool design pattern:

The object pool pattern is a software [creational design pattern](https://en.wikipedia.org/wiki/Creational_pattern) that uses a set of initialized [objects](https://en.wikipedia.org/wiki/Object_(computer_science)) kept ready to use – a "[pool](https://en.wikipedia.org/wiki/Pool_(computer_science))" – rather than allocating and destroying them on demand. A client of the pool will request an object from the pool and perform operations on the returned object. When the client has finished, it returns the object to the pool rather than [destroying it](https://en.wikipedia.org/wiki/Object_destruction); this can be done manually or automatically.

Object pools are primarily used for performance: in some circumstances, object pools significantly improve performance. Object pools complicate [object lifetime](https://en.wikipedia.org/wiki/Object_lifetime), as objects obtained from and returned to a pool are not actually created or destroyed at this time, and thus require care in implementation.

Implementation:

We make an array list called historyinfo by which we collect all the created object to use it again at different ways the only way to drop it is by clear() by which we

Restart the hole rogram

4-Singleton design pattern:

In [software engineering](https://en.wikipedia.org/wiki/Software_engineering), the singleton pattern is a [design pattern](https://en.wikipedia.org/wiki/Design_pattern_(computer_science)) that restricts the [instantiation](https://en.wikipedia.org/wiki/Instantiation_(computer_science)) of a [class](https://en.wikipedia.org/wiki/Class_(computer_programming)) to one [object](https://en.wikipedia.org/wiki/Object_(computer_science)). This is useful when exactly one object is needed to coordinate actions across the system. The concept is sometimes generalized to systems that operate more efficiently when only one object exists, or that restrict the instantiation to a certain number of objects. The term comes from the [mathematical concept of a singleton](https://en.wikipedia.org/wiki/Singleton_(mathematics)).

There are some who are critical of the singleton pattern and consider it to be an [anti-pattern](https://en.wikipedia.org/wiki/Anti-pattern) in that it is frequently used in scenarios where it is not beneficial, introduces unnecessary restrictions in situations where a sole instance of a class is not actually required, and introduces [global state](https://en.wikipedia.org/wiki/Global_variables) into an application.

Implementation: We used the singleton design pattern in both FillColor and BorderColor classes as they don’t have to be instantiated more than once.

5-Command design pattern:

In [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming), the command pattern is a [behavioral](https://en.wikipedia.org/wiki/Behavioral_Pattern) [design pattern](https://en.wikipedia.org/wiki/Design_pattern_(computer_science)) in which an object is used to [encapsulate](https://en.wikipedia.org/wiki/Information_Hiding) all information needed to perform an action or trigger an event at a later time. This information includes the method name, the object that owns the method and values for the method parameters.

Four terms always associated with the command pattern are command, receiver, invoker and client. A command object knows about receiver and invokes a method of the receiver. Values for parameters of the receiver method are stored in the command. The receiver then does the work. An invoker object knows how to execute a command, and optionally does bookkeeping about the command execution. The invoker does not know anything about a concrete command, it knows only about command interface. Both an invoker object and several command objects are held by a client object. The client decides which commands to execute at which points. To execute a command, it passes the command object to the invoker object.

Using command objects makes it easier to construct general components that need to delegate, sequence or execute method calls at a time of their choosing without the need to know the class of the method or the method parameters. Using an invoker object allows bookkeeping about command executions to be conveniently performed, as well as implementing different modes for commands, which are managed by the invoker object, without the need for the client to be aware of the existence of bookkeeping or modes.

Implementation:

We make a command interface which contain undo redo function && getval () which we use it in order to undo and redo all the valid operation

In order to be capable to undo the whole operation

And the command line container by which we will could be undo any type of object

And command manager which which return the command to the command line container.

6-Model-View-Controller(MVC) design pattern:

Model–view–controller (MVC) is a software [architectural pattern](https://en.wikipedia.org/wiki/Architectural_pattern) for implementing [user interfaces](https://en.wikipedia.org/wiki/User_interface) on computers. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.

Traditionally used for desktop [graphical user interfaces](https://en.wikipedia.org/wiki/Graphical_user_interface) (GUIs), this architecture has become extremely popular for designing [web applications](https://en.wikipedia.org/wiki/Web_application).

Implementation: We created a ‘View’ class which is “Main.fxml” which contains the implementation of user interface (GUI) , ‘Controller’ class which is “Main” class which its role is to link between the classes containing the logic code and the ‘View’ class , and finally the ‘Model’ classes which are the rest of classes.

**Basic OOP Concepts used:**

1-Inheritance:

Used to implement the Decorator design pattern at which an abstract super class “ShapeDecorator” is created and 2 concrete classes “FillColor” and “BorderColor” extend it.

2-Encapsulation:

All the fields in all the classes are private and are accessed only through setters and getters.

3-Polymorphism:

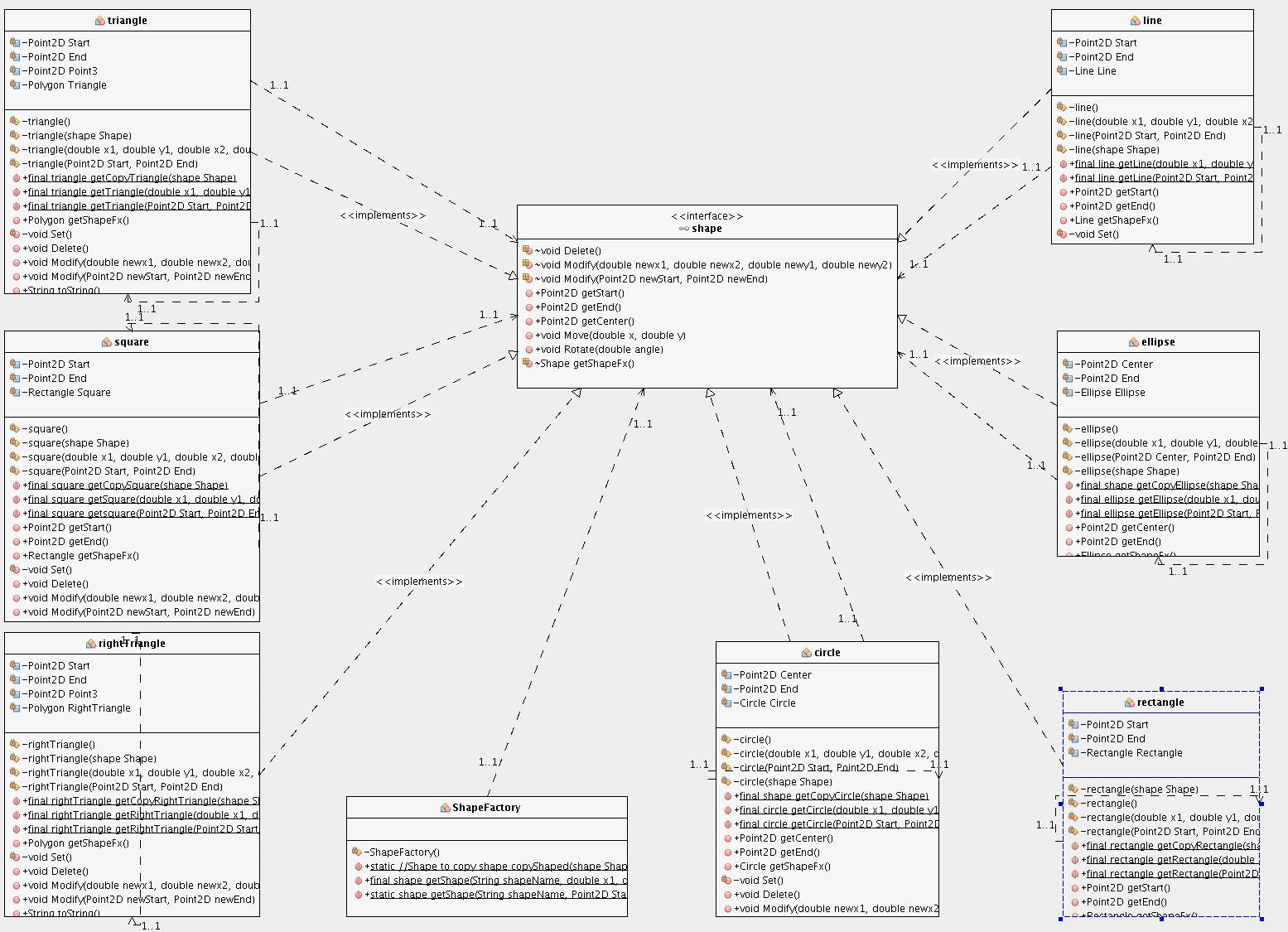
All the shapes are instantiated polymorphically through the “shape” interface for the creation of object is dynamic.

4-Abstraction:

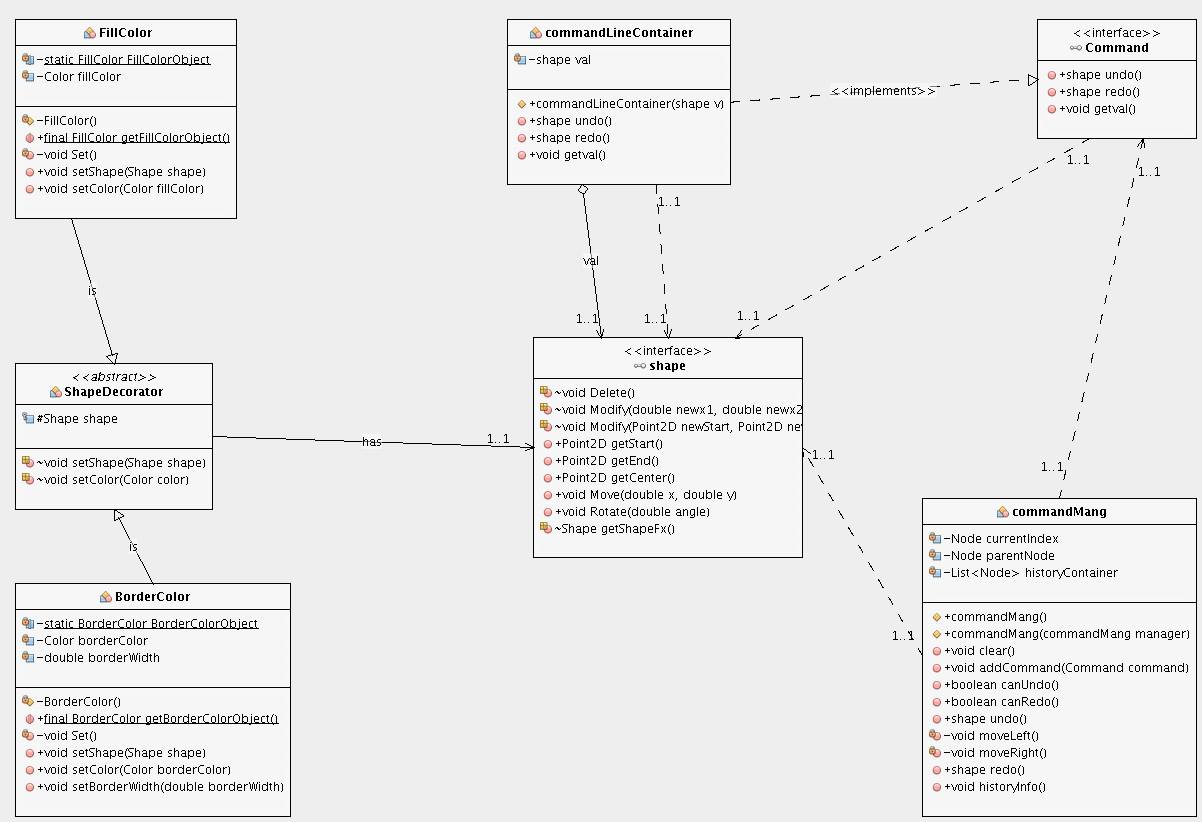
The user uses the methods without knowing their implementation.

**UML Design:**

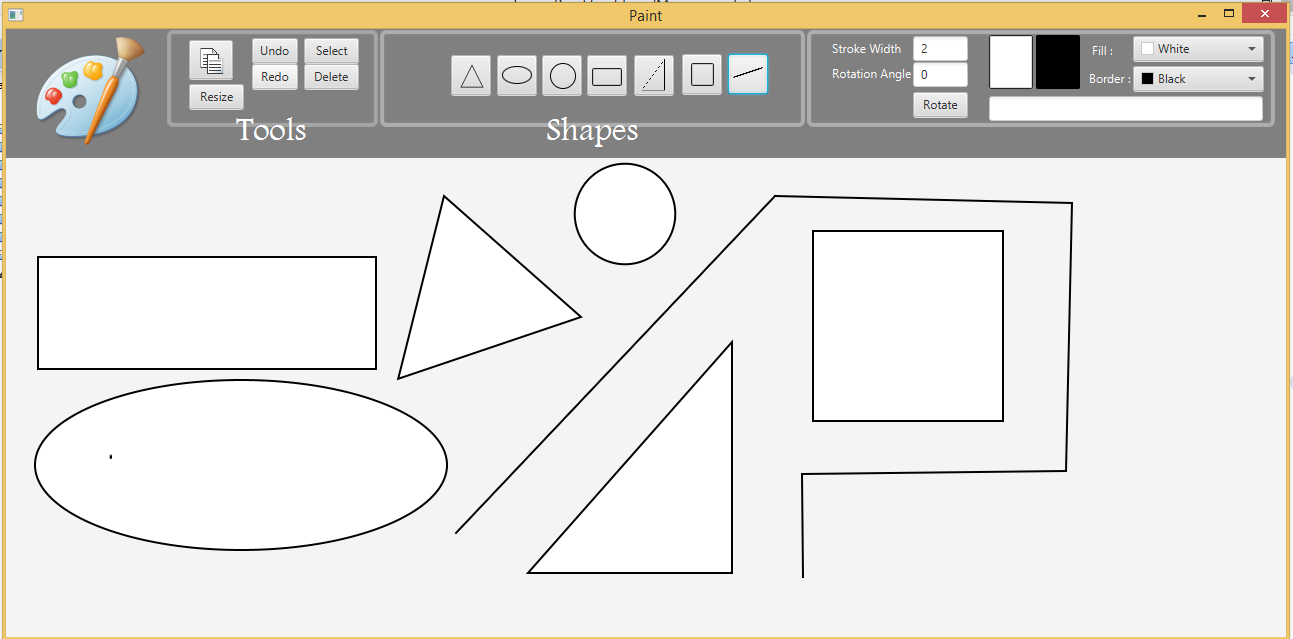
1-“shape” interface relationship with all the shapes and “ShapeFactory:

****

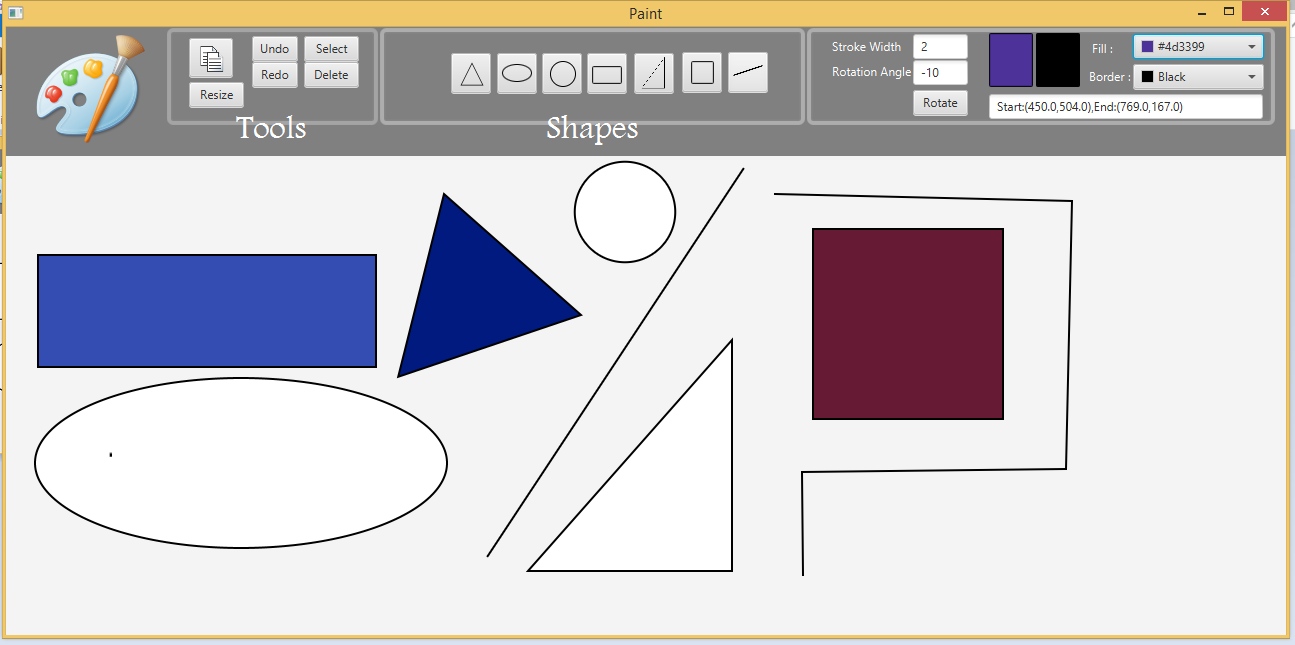
2-“shape” interface relationship with “Command” interface and “ShapeDecorator” abstract class:



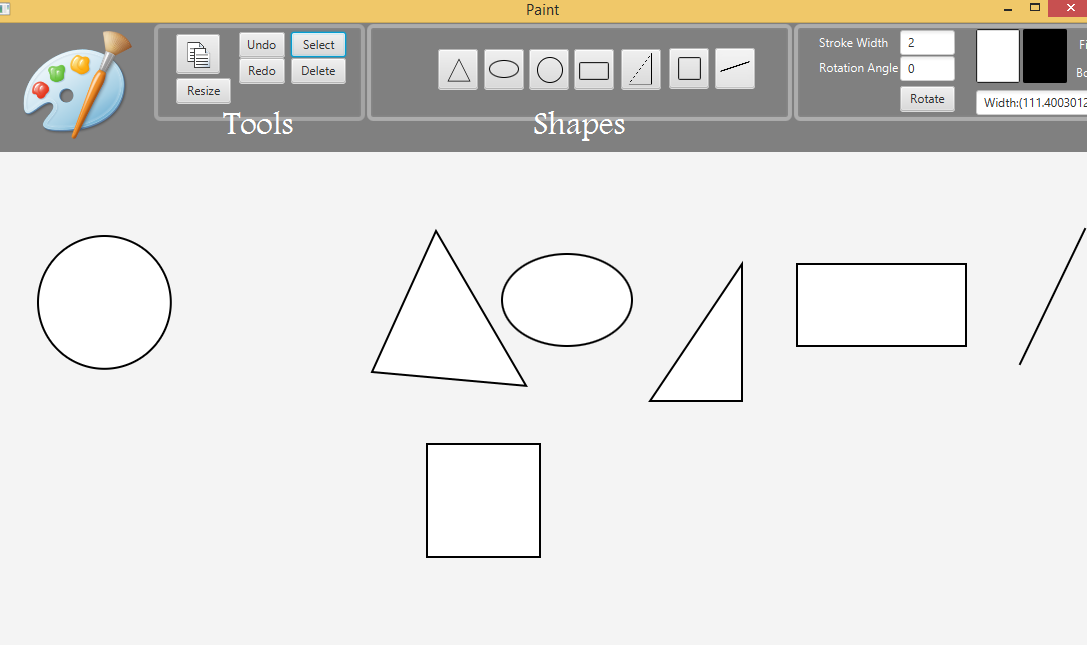
**Snapshots from the program:**



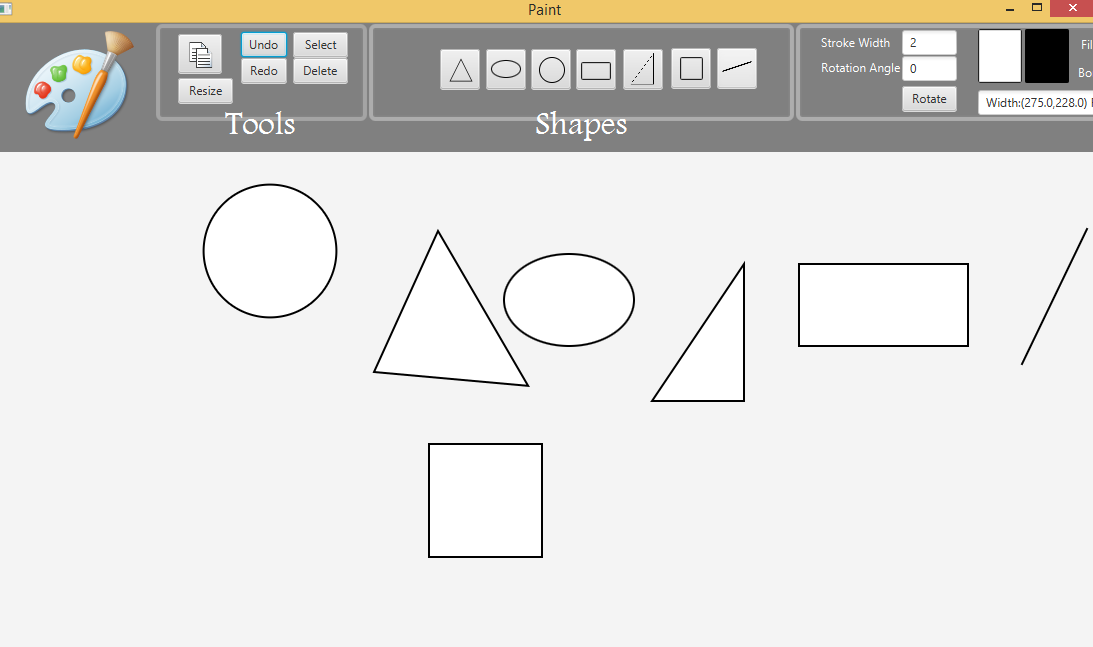
In the current Figure it shows all the available shapes to draw .



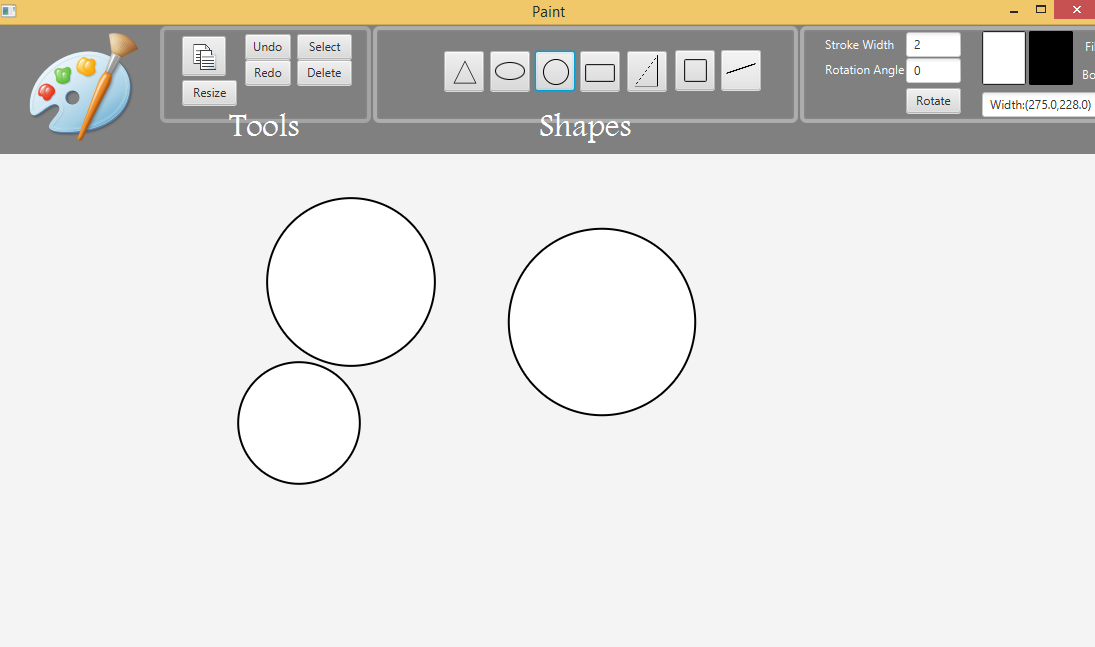
This figure shows the ability to color a shape after or while drawing it .



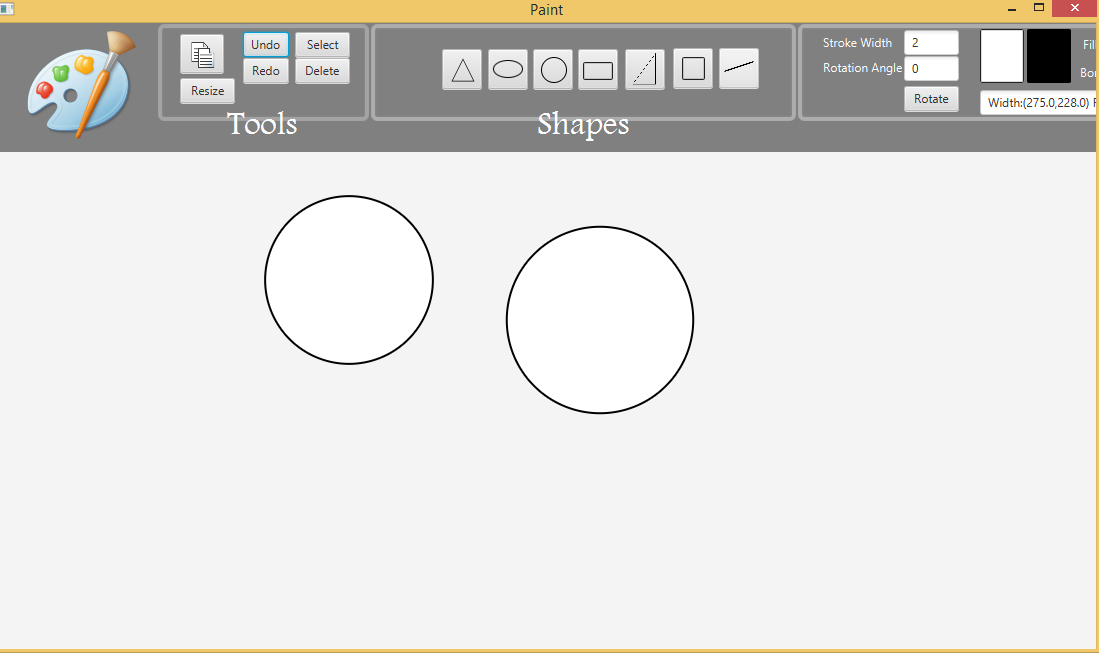
In this figure we moved the circle a little to the left



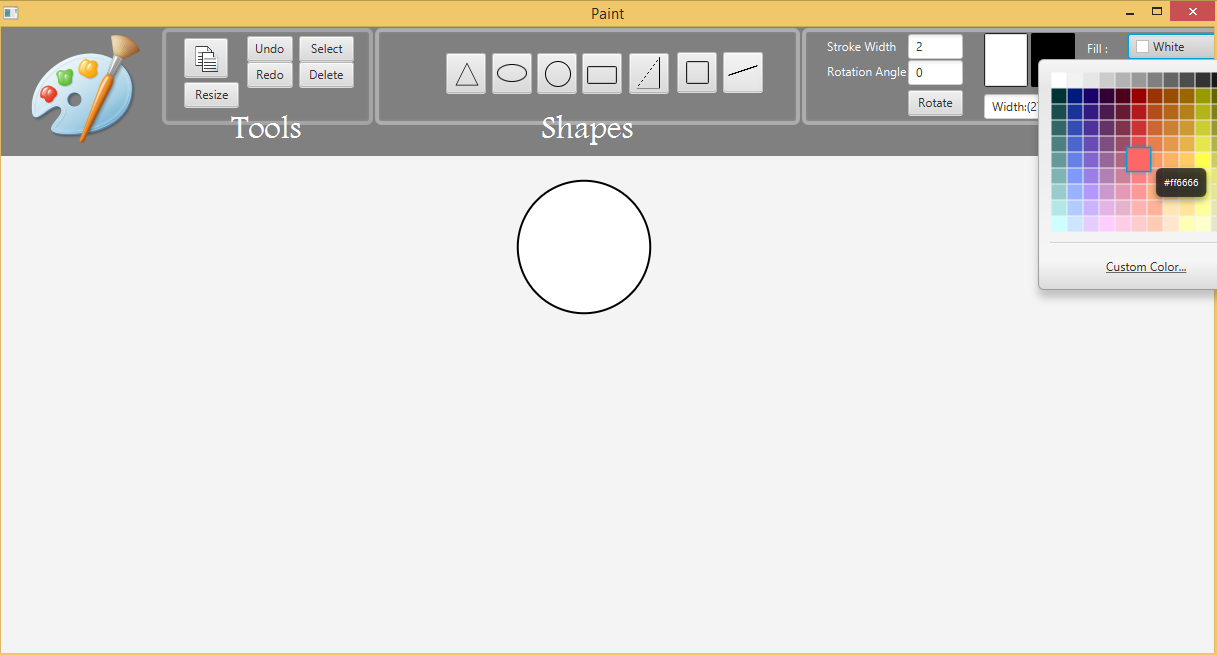
This figure shows the redo of moving the circle



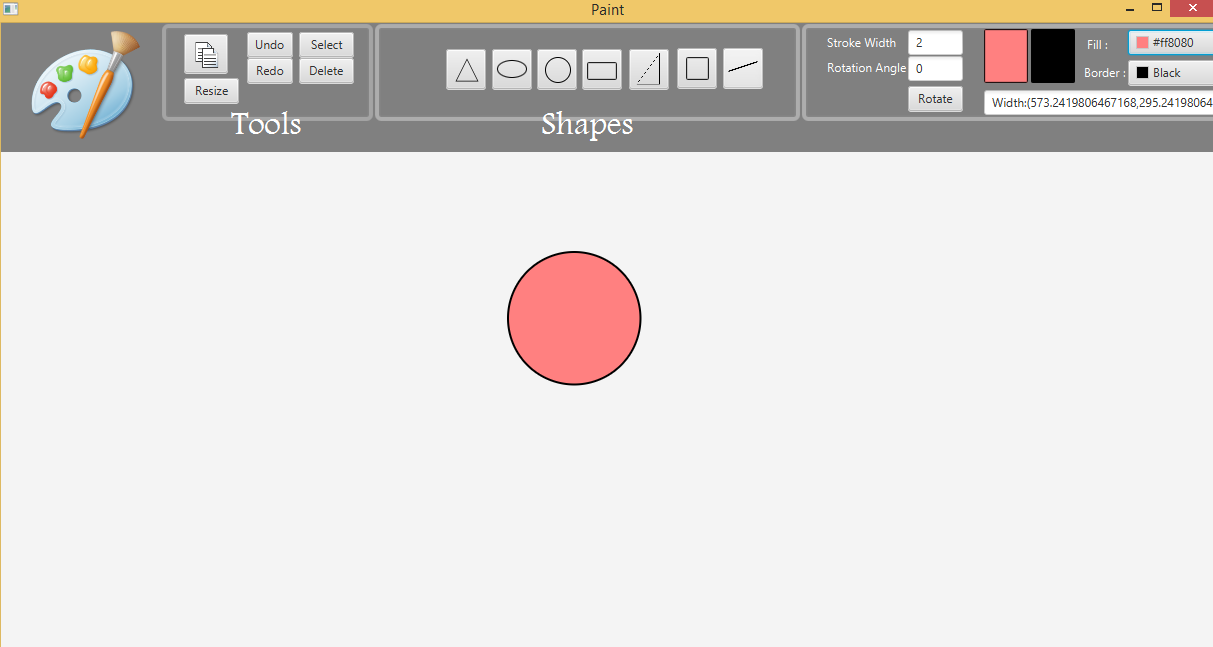
In this figure we created 3 new circles



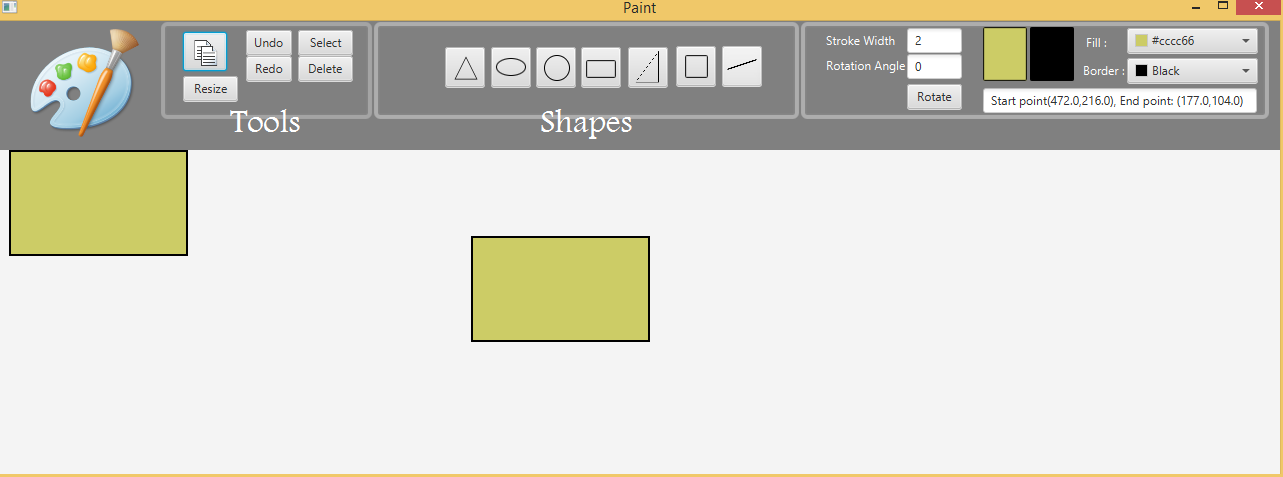
In this figure shows the last circle drawn was removed after undo.



This figure shows the Color picker and choosing the fill or the border color of the selected shape.



This figure shows the shape after filling it with the chosen color from the color picker.



This figure shows the property of copying a shape and it puts it to the corner of the panel.

**User guide:**

-Select a shape icon to draw the selected shape.

-Select a border color and/or a fill color from the border color icon and the fill color icon before drawing a shape to add this color to it.

-Specify a desired stroke width before drawing a shape.

-Tap on the select icon and then select on a desired drawn shape and this permits you to:

1-Add currently selected fill color to this shape.

2-Add currently selected border color to this shape.

3-Make a copy of this shape at fixed dynamic location by taping on the copy icon.

4-Move this shape by dragging this shape.

5-Resize this shape by selecting the resize icon then by dragging this shape.

6-Delete this shape by selecting the delete icon.

7-Rotate this shape by selecting the desired angle of rotation.

-Undo all the previous actions by selecting the undo icon.

-Redo all the undone actions by selecting the redo icon.