Marron Hawas n9667024 Christopher Linnan n8845441

Team Number: 71

Assignment 2: Vector Design Tool

CAB302 Software Development

Tutor: Dilesha Dissanayake Wasala

Tutorial Time: Monday, 4-6pm GP-S518

Due Date: 31st May 2019

Contents

[Statement of completeness 3](#_Toc10385292)

[Statement of contribution 3](#_Toc10385293)

[Agile software development 3](#_Toc10385294)

[Software architecture 4](#_Toc10385295)

[Package Hierarchies: 4](#_Toc10385296)

[Class Hierarchy 5](#_Toc10385297)

[Class Summary 6](#_Toc10385298)

[*McCanvas* 6](#_Toc10385299)

[*McShape* 7](#_Toc10385300)

[*VecFile* 7](#_Toc10385301)

[*VecFileFilter* 7](#_Toc10385302)

[*Polygon* 7](#_Toc10385303)

[*Ellipse* 8](#_Toc10385304)

[*Rectangle* 8](#_Toc10385305)

[*Plot* 8](#_Toc10385306)

[Object Oriented Programming 9](#_Toc10385307)

[Manual 12](#_Toc10385308)

The purpose of this assignment is to develop a Java program for drawing vector-based graphics, which makes use of the proprietary design language called VEC which is not used by any other hardware or software on the market. Using these VEC files, the “Vector Design Tool” software created by Team 71, can load the values found in these files to generate vector images or a user may create vector images and save them into a VEC file. The “Vector Design Tool” is a user friendly and intuitively designed piece of software, meant for the use of any user, of any technical level of experience.

# Statement of completeness

All required core functionality completed. Team 71 contains two team members, the additional functionality added was the “Zoom” function which was implemented using a slider.

# Statement of contribution

All code creation was shared, Mr Hawas was prominent with the coding of the GUI, basic shape button functionality and testing. Mr Linnan was prominent in the coding of additional functions such as “Save”, “Load”, “Zoom”, “Fill” and “Edge” colour selection as well as refactoring and code quality and conventions. Both team members worked on the report.

# Agile software development

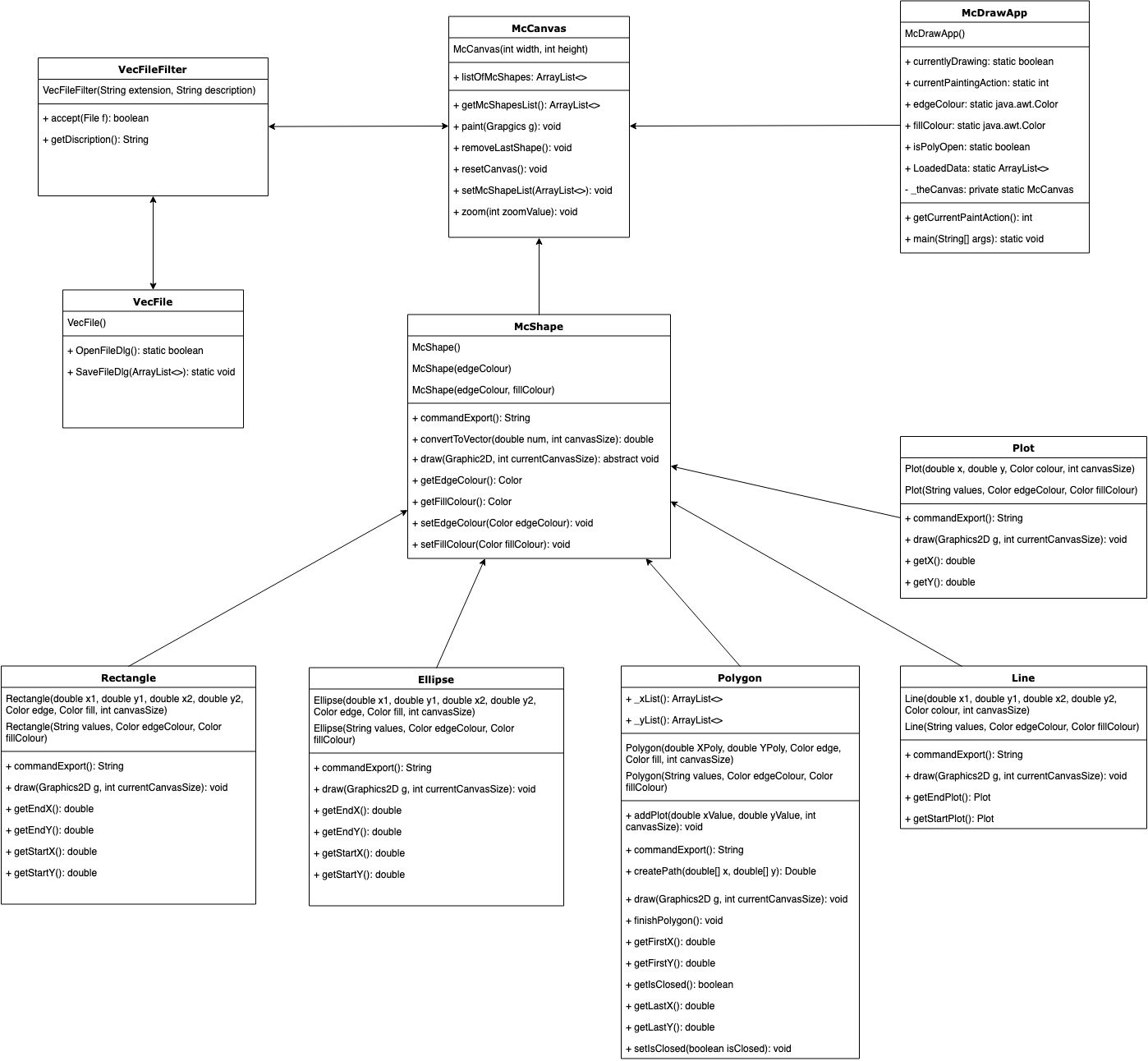
To create the vector drawing application Team 71 used Agile Methodologies as the bases of the software development cycle. Agile projects work from user stories rather than detailed design documents. Initially the requirements for the project were broken down into user stores which could then be implemented. Another core aspect of agile projects is rapid development of working software with continuous refinements. This was the cornerstone of our software development process as we went through an iterative process of rapidly developing functionality, followed by refinement, and implementation of automated tests.

Initially Mr Hawas built the GUI that gave the general layout and allowed for drawing of basic shapes. Throughout the development cycle as user stories were implemented into the application the GUI was updated with buttons, sliders, and menu items to facilitate the newly added features. After each update to the application the code was refactored to create cleaner code, fix any abnormalities, and to improve the applications architecture.

After the initial implementation tests were written by Mr Hawas for the features that had been implemented. Following this, after each new feature Mr Hawas created tests for each newly added feature.

After each substantial commit the team discussed, via the internet, what had been done, how it had been implemented and which features to work on. Each week there was a face to face meeting to reflect on how the project was going and any challenges each member was facing. As this was an agile driven project, only working versions of the software were ever pushed to the repository.

# Software architecture

****

# Package Hierarchies:

* files
* gui
* shapes
* tests

# Class Hierarchy

* java.lang.Object
  + java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - java.awt.Container
      * javax.swing.JComponent (implements java.io.Serializable)
        + gui.McCanvas
      * java.awt.Window (implements javax.accessibility.Accessible)
        + java.awt.Frame (implements java.awt.MenuContainer)

javax.swing.JFrame (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants)

gui.McDrawApp

* javax.swing.filechooser.FileFilter
  + files.VecFileFilter
* shapes.McShape
  + shapes.Ellipse
  + shapes.Line
  + shapes.Plot
  + shapes.Polygon
  + shapes.Rectangle
* tests.TestEllipse
* tests.TestLine
* tests.TestMcCanvas
* tests.TestPlot
* tests.TestPolygon
* tests.TestRectangle
* tests.TestVecFile
* tests.TestVecFileFilter
* files.VecFile

# Class Summary

#### McDrawApp

The McDrawApp is the root pane container. Here the GUI is created, to which the “McCanvas” component is placed and the McShape imported. This class extends the JFrame class which in turn extends the java.awt.Frame class. The initial extension to JFrame allows for Swing component architecture which is required for drawing. The subsequent extension to become a McDrawApp allows for custom properties and methods required for the VEC drawing application such as properties to keep track of what drawing action is currently selected, the current chosen colours, and the canvas which is being drawn upon. The class initialises the GUI elements and displays the window. Users interact with this and the canvas component when using the application. This class allows the user to open and save VEC files. Both functions interact with the McCanvas and VecFile classes to perform their respective function. Load uses the VecFile to parse the contents of a VEC file which uses the commands and parameters to create a new list of shapes which is then passed by the McDrawApp class to its child McCanvas class. The Save function retrieves a list of shapes from the McCanvas and passes it to the VecFile class. The VecFile class iterates through each shape constructing a new VEC file based on each shapes output.

## McCanvas

The McCanvas component is a blank square area of the screen onto which the user can draw shapes. The class extends JComponent and is a child of the McDr**a**wApp class which is a top-level Swing container. The extension from the component class means it can support painting and evens. The McCanvas adds functionality specific to the drawing app by providing properties and functions unique to the application. This includes the list of custom shapes, and helper properties that record X,Y coordinates. The class contains numerous functions related to drawing shapes to the screen and its properties e.g. the list of shapes. The instantiation of the class initialises action listeners which implement the user interaction logic. This user logic interacts with the various shape classes by way of adding new shapes to the list being stored on the canvas. The canvas calls each shape within its list of shapes to draw themselves to the canvas. The class also allows other classes to access its list of shapes via getting methods.

## McShape

The McShape class is an abstract class which is extended for each custom shape class. It provides basic logic for getting and setting colours along with the conversion from absolution XY coordinates to relative coordinates used in vector drawings. It provides two abstract methods which must be implemented. These are the draw function and export function which are used by the McCanvas for drawing the particular shape to the canvas and the VecFile class for converting the individual shape into a string when being saved to a VEC file.

## VecFile

A custom file format for storing information about a drawing made in the McDrawApp painting application. The VEC file format consists of strings which begin with a command and end with parameters for the given command. Each line within the file is a new command with adjacent parameters. The commands and parameters are parsed when a vec file is being opened by the McDrawApp class.

## VecFileFilter

A custom file format filter for the .VEC file format. This class is used by the VecFile class within the open and save dialogues. It filters out all other files besides those that have the .vec extension.

## Polygon

The Polygon extends the abstract shape class McShape to give more customisation and control over the object. The Polygon class has two constructors, one used by the McCanvas class when shapes are created by the user and the other constructor is used by the VecFile class when opening saves files. For painting to the McCanvas the Polygon class creates a java.awt.geom.Path2D based on the internal array of points. The Path2D object is used within the shapes draw function which is called by McCanvas.

## Ellipse

The Ellipse extends the abstract shape class McShape to give more customisation and control over the object.The Ellipse class has two constructors, one used by the McCanvas class when shapes are created by the user and the other constructor is used by the VecFile class when opening saves files. For painting to the McCanvas the Ellipse class creates a java.awt.geom.Ellipse2D based on the four internal location properties. The Ellipse2D object is used within the shapes draw function which is called by McCanvas.

## Rectangle

The Rectangle extends the abstract shape class McShape to give more customisation and control over the object. The Rectangle class has two constructors, one used by the McCanvas class when shapes are created by the user and the other constructor is used by the VecFile class when opening saves files. For painting to the McCanvas the Rectangle class creates a java.awt.geom.Rectangle2D based on the four internal location properties. The Rectangle2D object is used within the shapes draw function which is called by McCanvas.

## Plot

The Plot extends the abstract shape class McShape to give more customisation and control over the object. The Plot class has two constructors, one used by the McCanvas class when shapes are created by the user and the other constructor is used by the VecFile class when opening saves files. For painting to the McCanvas the Plot class creates a zero length java.awt.geom.Line2D based on the two internal location properties. The Line2D object is used within the shapes draw function which is called by McCanvas.

Line

The Line extends the abstract shape class McShape to give more customisation and control over the object. The Line class has two constructors, one used by the McCanvas class when shapes are created by the user and the other constructor is used by the VecFile class when opening saves files. For painting to the McCanvas the Line class creates a java.awt.geom.Line2D based on the two internal properties which hold the start and end locations of the Line as Plot objects. The Line2D object is used within the shapes draw function which is called by McCanvas

TestClasses

Each test class implements several test cases that test all functionality of the class for which it is named. The test cases cover 100% of the classes, 100% of the methods and 93% of the lines of code. The Tests interact with the other classes by instantiating an object of that class then checking the objects properties against an expected result. Often the values checked will be retrieved by use of the particular objects accessor methods.

# Object Oriented Programming

During the construction of this software, Team 71 utilised advanced Object Oriented Programming principles. This included method concepts of: Abstraction, Encapsulation, Inheritance and Polymorphism. Below are some examples of these principles used in the “Vector Design Tool” software:

Abstraction

Applying abstraction means that each object should only expose a high-level mechanism for using it. This mechanism should hide internal implementation details. It should only reveal operations relevant for the function of the software. As an example, in the “Vector Design Tool” project, the “Save as” option requires the program to use multiply methods, classes and parsing of data before the the “Save as” task is complete, without the users knowledge or understanding. Changes to the project, such as adding new shape classes or different drawing styles, would rarely affect the “Save as” functionality. Additionally, abstraction was used in the class design for shapes.

The McShape class is an abstract class which is the simplest idea of a shape. It contains common methods and properties that all other derived shapes are expected to have. The exposed functions within the shape classes abstract away the internal details of the class. For example, each Shape has a unique constructor that takes variables as parameters and processes the data to create the shape. The detail of shape creation is abstracted away and simplified to only the input parameters as objects outside of a particular shape do not need to know the process by which those values determine the shapes properties.

Encapsulation

Encapsulation is achieved when each object keeps its state private, inside a class. Other objects don’t have direct access to this state. Instead, they can only call a list of public methods. As an example, in the “Vector Design Tool” project, encapsulation is used to generate all the shapes onto the “McShape” Class. Each shape type has its own class (Plot, Line, etc.), which are defined by the input from the mouse controlled by the user and are created with the “draw()” method. Each object shape is then stored in “listOfMcShapes” Array List to be utilised by the “McCanvas” Class.

Additionally, each Shape class uses encapsulation. A shape has private data members, such as its edge colour or fill colour, which can be accessed through public methods (getEdgeColour, setEdgeColour). These methods restrict how other objects can interact with a given shape. Validation, parsing, and formatting can occur in these methods to ensure that the object's internal state is always valid.

Inheritance

Due to shared common logic, objects are often very similar. We reuse the common logic and extract unique logic into a separate class by adopting inheritance principals. This means that you create a child class, by deriving from other parent classes, to form a class hierarchy. As an example, in the “Vector Design Tool” project, “McShape” is an abstract parent Class from which all other Shapes are derived. As the concrete shape classes of Line and Rectangle derive from the parent McShape class they inherit the properties and methods of the parent class. The parent class also provides abstract methods which each of the child classes must implement as the method is required by all classes, but its implementation is unique to each shape as can be seen with the shapes Draw function and Export functions.

Polymorphism

Polymorphism is the principle that provides the ability of a class to have multiple implementations and functions with the same name. As an example, in the “McCanvas” class, when the “paint” method is called:

for (McShape s : listOfMcShapes){  
 s.draw(graphicSettings, this.getHeight());  
}

Here the same method is called, the draw() method, on different classes that derive from the McShape class. Each derived class has overwritten the abstract method from the parent class with its own unique implementation. This particular example is known as Dynamic Polymorphism as opposed to Static Polymorphism. An example of Static Polymorphism used in this project, in the “McCanvas” class occurs here:

case 1:  
 myTempShape = drawPoint(drawStart.getX(), drawStart.getY());  
 break;  
case 2:  
 myTempShape = drawLine(drawStart.getX(), drawStart.getY(), drawEnd.getX(), drawEnd.getY());  
 break;

Here “.getX()” and “.getY()” methods are used on different objects from different classes.

# Manual

The “Vector Design Tool” software consists of a visual based, user friendly, Graphical User Interface. Developed using Java, with a Software Development Kit(SDK) version 11, for installation on Windows operating systems.

**Be sure to utilise the mouse over buttons hover tooltips for quick assistance.**

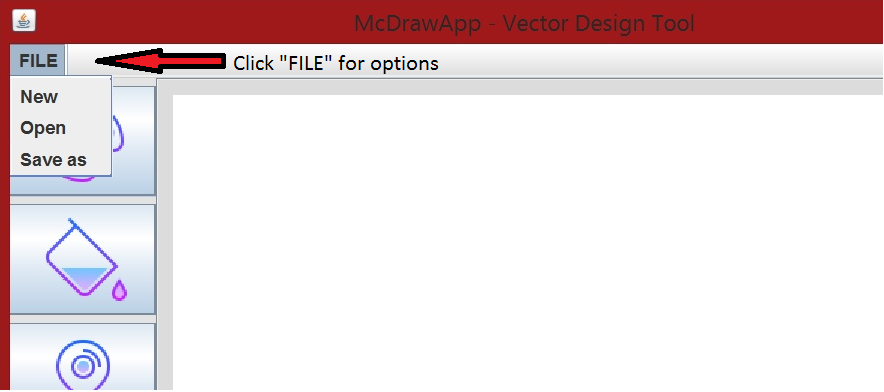
A brief description of all the buttons and functions may be found below:

**FILE:** Clicking on the file button displays these three(3) options:

**(1) New:** Selecting this option will clear all objects and shapes on the drawing pane.

**(2) Load:** Selecting this option will provide you with the option to select a .VEC extension file

**(3) Save as:** Selecting this option will provide you the opportunity to save your current work under the filename of your choosing. The .VEC extension will automatically be placed on your file name, you do not need to do it manually.

****

***Figure: Location of “File” options.***

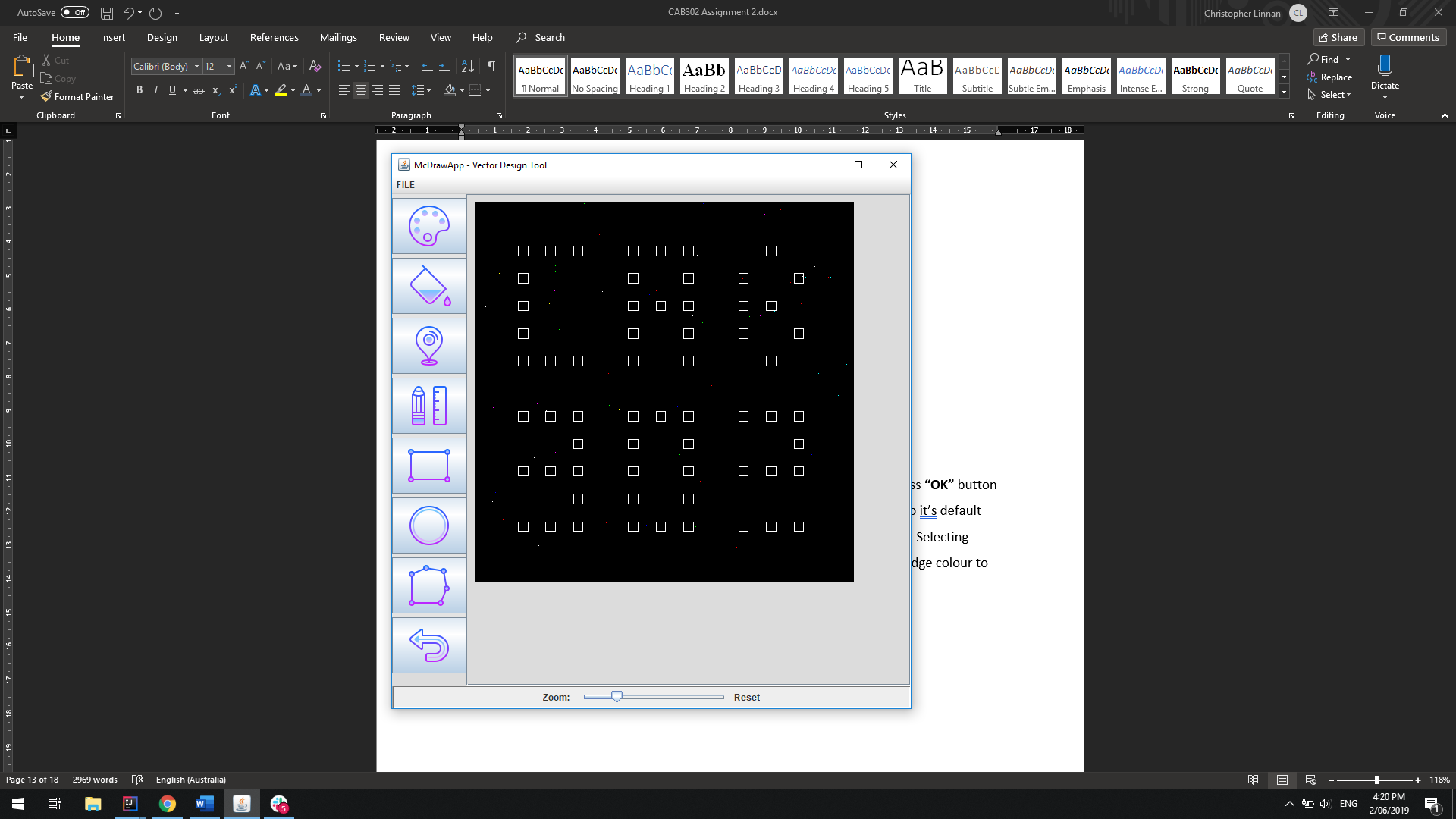
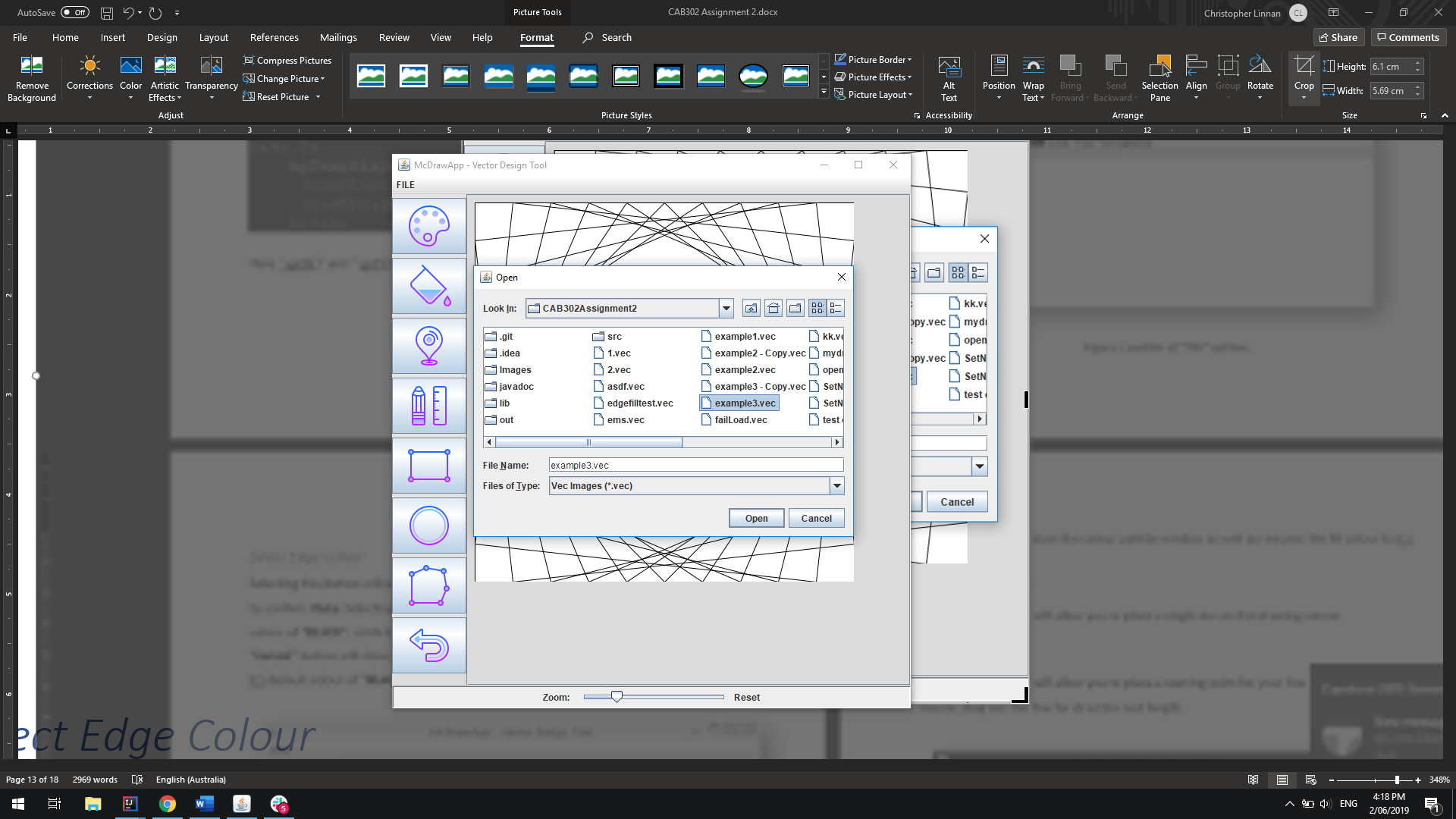
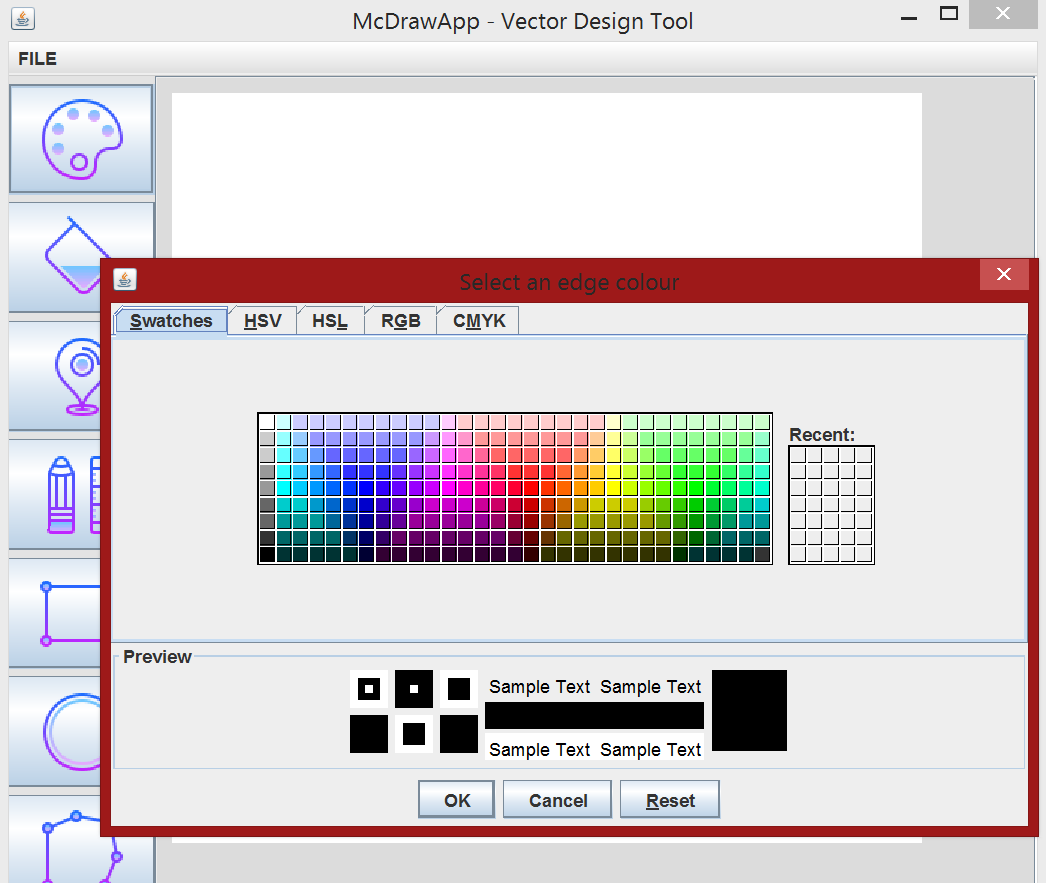


Figure: Open file dialogue Figure: The new file is opened

Select Edge ColourSelecting this button will prompt you to select a colour for the edge lines, press **“OK”** button to confirm. **Note:** Selecting the **“Reset”** button will restores the edge colour to it’s default colour of **“BLACK”**, while keeping the colour palette window open. **Also Note:** Selecting **“Cancel”** button will close the colour palette window as well as restores the edge colour to it’s default colour of **“BLACK”.**

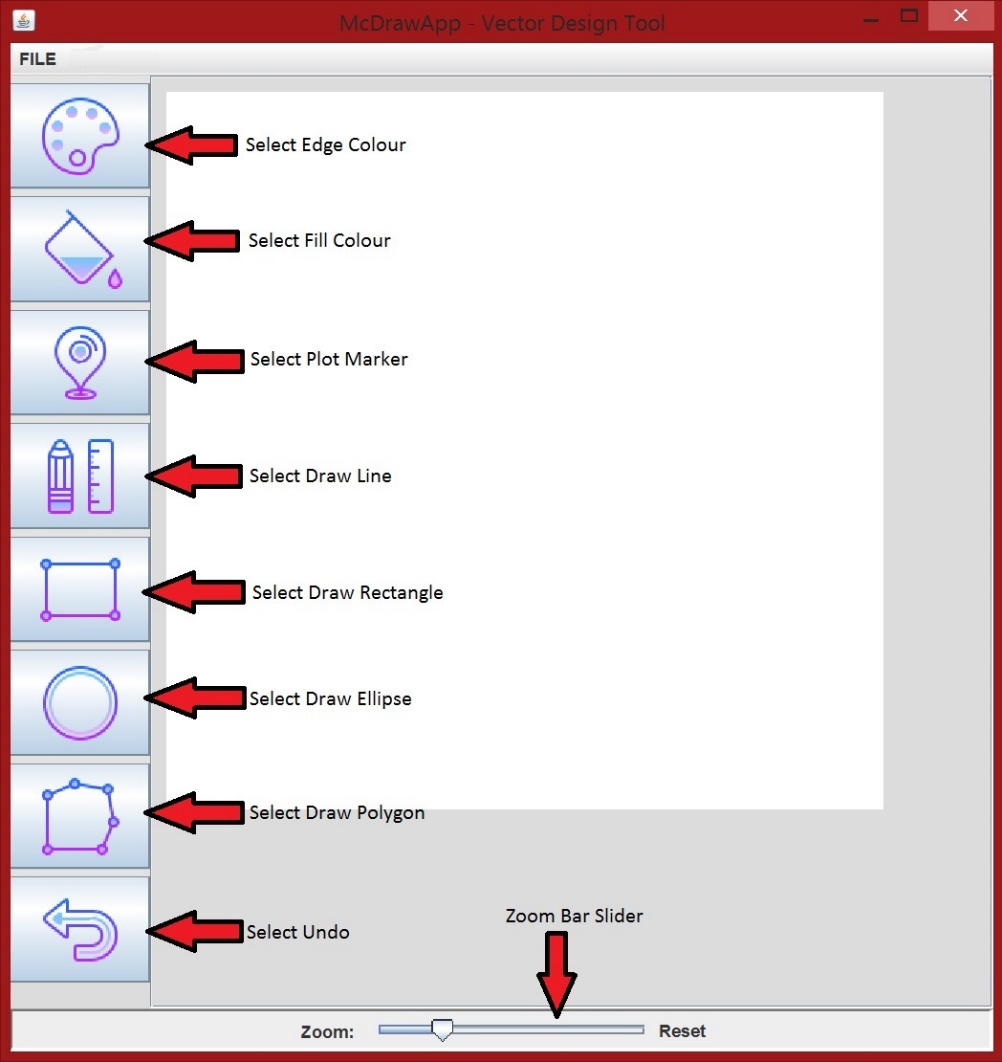
Select Fill ColourSelecting this button will prompt you to select a colour for the filling of shapes, press **“OK”** button to confirm. **Note:** Selecting the **“Reset”** button will restores the fill colour to it’s default of **“NONE”**, while keeping the colour palette window open. Also **Note:** Selecting the **“Cancel”** button will close the colour palette window as well as restores the fill colour to it’s default of **“NONE”.**

****

***Figure: Colour Palette Window***

Select Plot MarkerSelecting this button will allow you to place a single dot on the drawing canvas.

Select Draw LineSelecting this button will allow you to place a starting point for your line and using the mouse, drag out the line for direction and length.

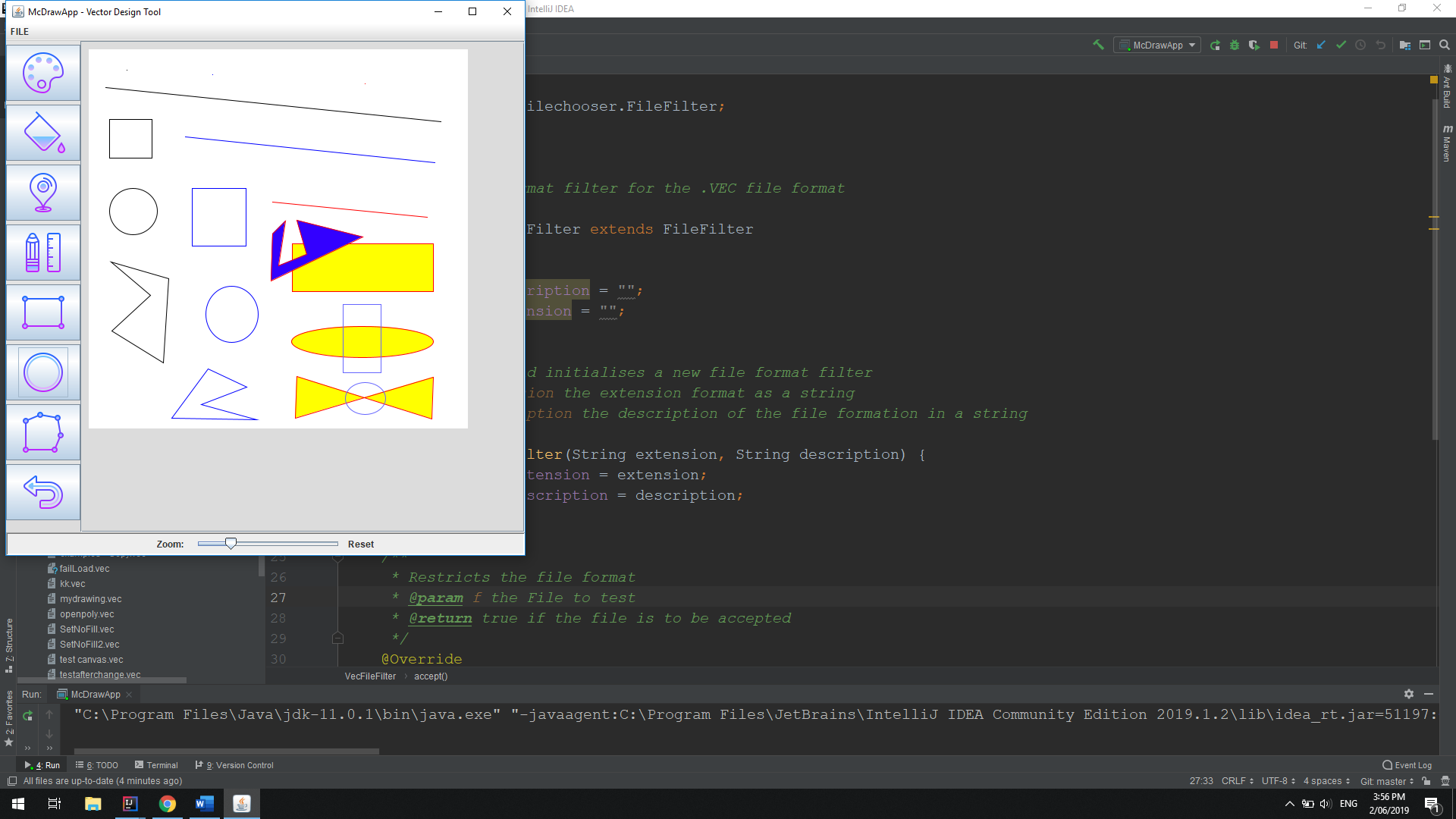
****

***Figure: Location of Vector Design Tool buttons***

Select Draw RectangleSelecting this button will allow you to place a starting point for your rectangle and using the mouse, drag out the rectangle for width and height.

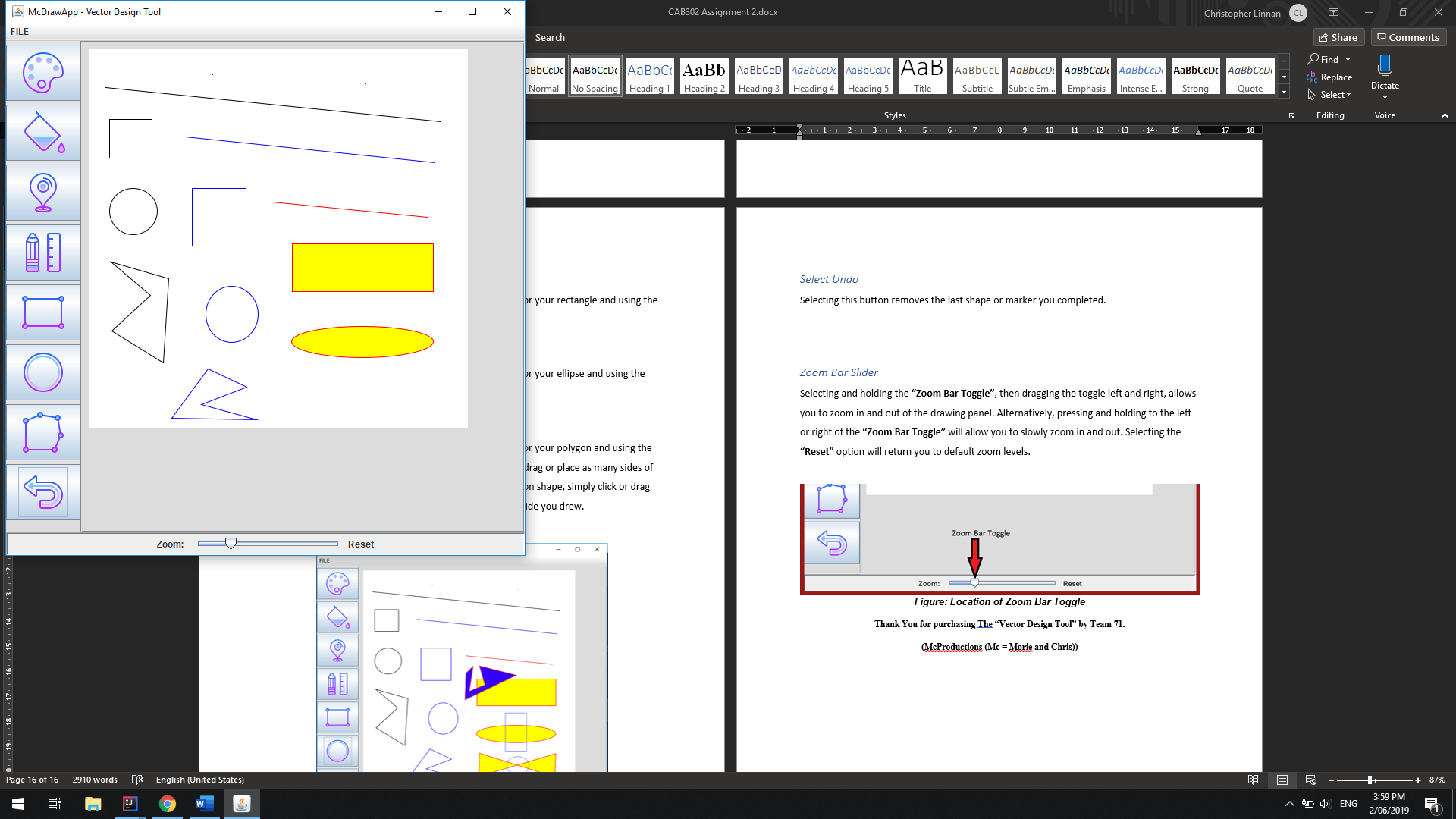
Select Draw EllipseSelecting this button will allow you to place a starting point for your ellipse and using the mouse, drag out the ellipse for width, height and shape.

Select Draw PolygonSelecting this button will allow you to place a starting point for your polygon and using the mouse, drag out the first line of your polygon. You may now drag or place as many sides of the polygon shape as you like. **Note:** To complete your polygon shape, simply click or drag your last side of the polygon near or on the start of the first side you drew.



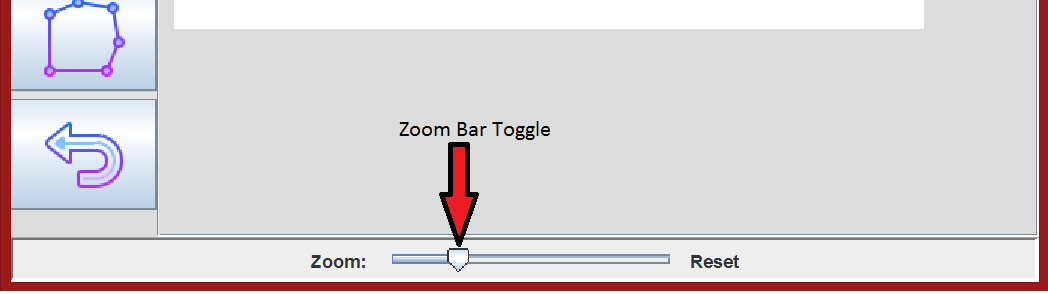
**Figure: Image showing all shapes and functionality**

Select UndoSelecting this button removes the last shape or marker you completed.



**Figure: Undo pressed four times to remove the last 4 items**

Zoom Bar SliderSelecting and holding the **“Zoom Bar Toggle”**, then dragging the toggle left and right, allows you to zoom in and out of the drawing panel. Alternatively, pressing and holding to the left or right of the **“Zoom Bar Toggle”** will allow you to slowly zoom in and out. Selecting the **“Reset”** option will return you to default zoom levels.

****

***Figure: Location of Zoom Bar Toggle***

## 

**Figure: Minimum level of zoom**

## 

**Figure: Maximum level of zoom**

## 

## Thank You for purchasing The “Vector Design Tool” by Team 71.

## (McProductions (Mc = Marron & Chris))