CAB302 A2

Alexander Brimblecombe
N10009833

Group 471

Contents

Statement of Completeness

All the basic functionality has been implemented in the program.

There is an extended feature which allows the user to export the canvas to PNG. However size choosing was not implemented for this – it exports with constant dimensions. This is because the project was completed in a group of one person, and the additional functionality wasn't a requirement.

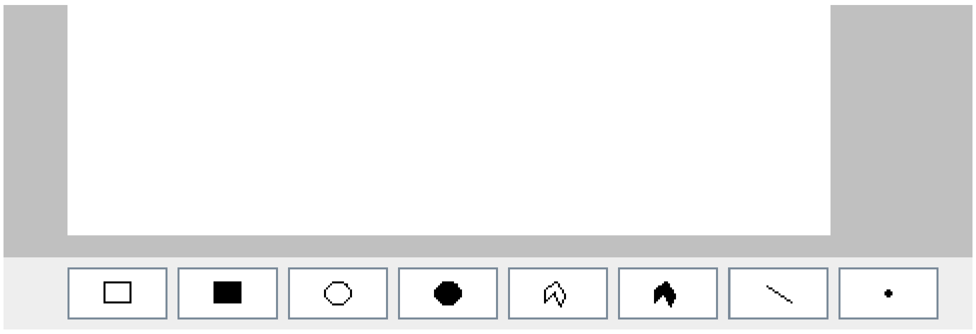
Contribution

Completed by a group of one, so there was no work to divide.

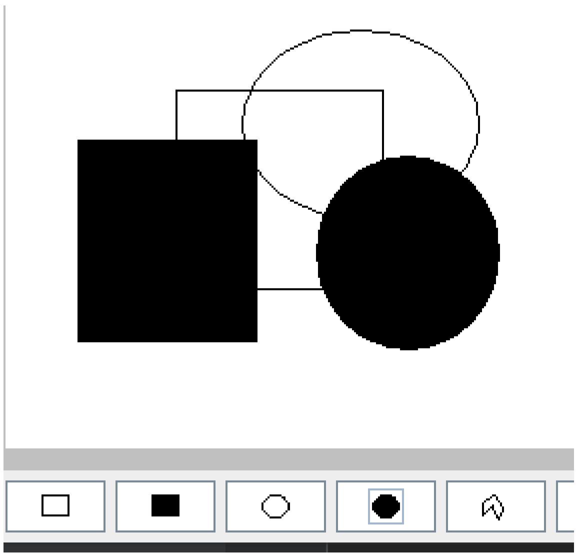
Instructions

Drawing

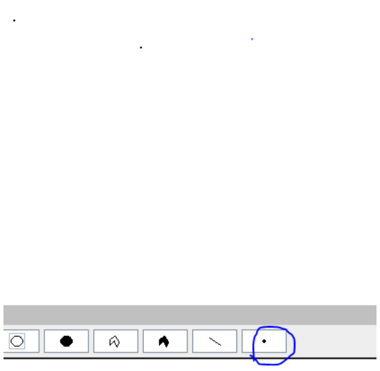
Select drawing commands from the bottom panel. Solid icons denote a filled shaped.



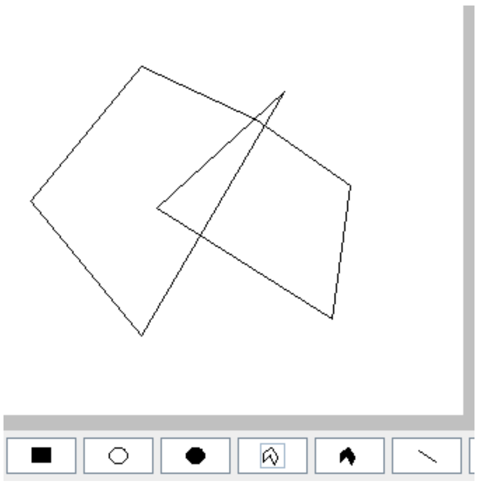
For rectangles, ellipses and lines **left click** to start drawing. **Drag** until the point you desire, then **Release** to commit.



For <u>plots</u> (single point) **left click** the location to immediately commit the instruction.



For <u>polygon</u> use **left click** to start drawing the shape, add new points by **left clicking**. After at least one point is added **right click** to finish drawing.

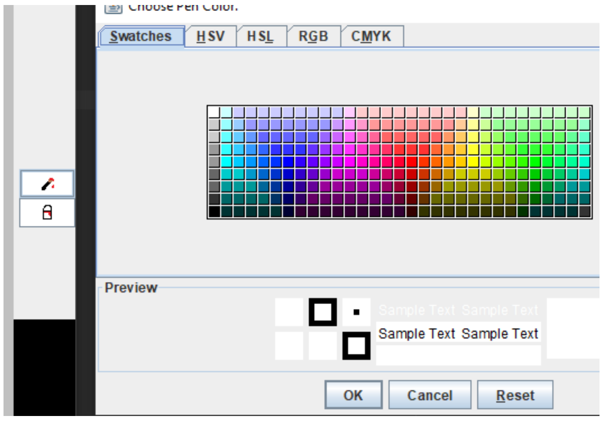


Colours

To select both **pen** colour and the **fill** colour use the buttons on the right-hand panel. These will open a colour chooser.

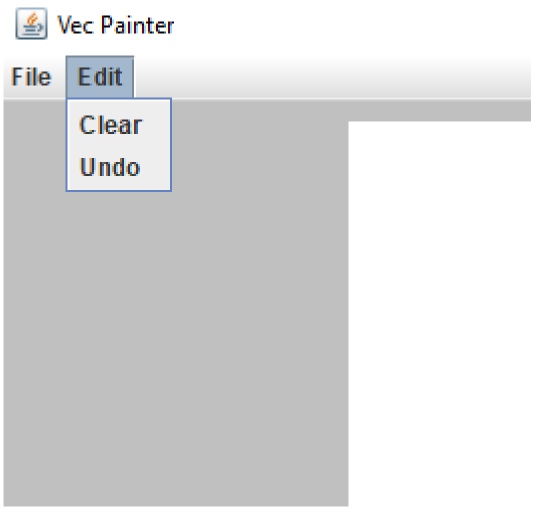
Use the paint brush icon to open a chooser for the **pen** color.

Use the paint bucket icon open a chooser for the **fill** color.



Undo / Clear

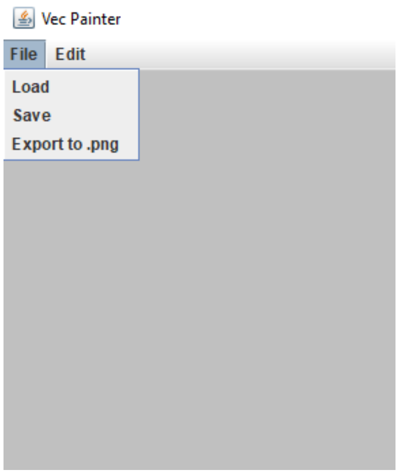
To undo the last command press **ctrl z** on the keyboard or navigate to the **edit** menu on the top panel – click undo.



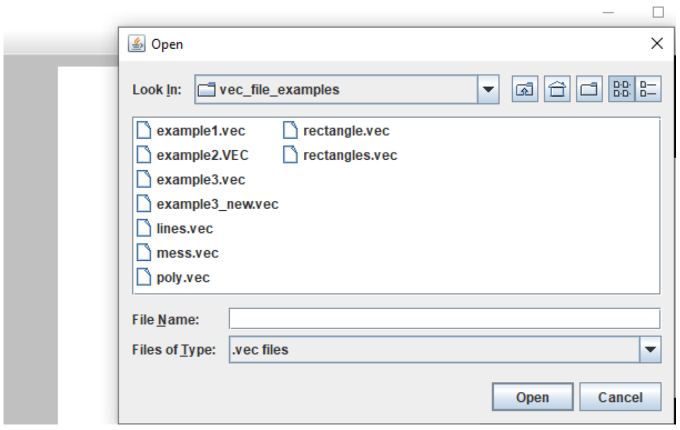
The menu also provides the option to clear the screen. This will remove everything from the canvas.

I/O

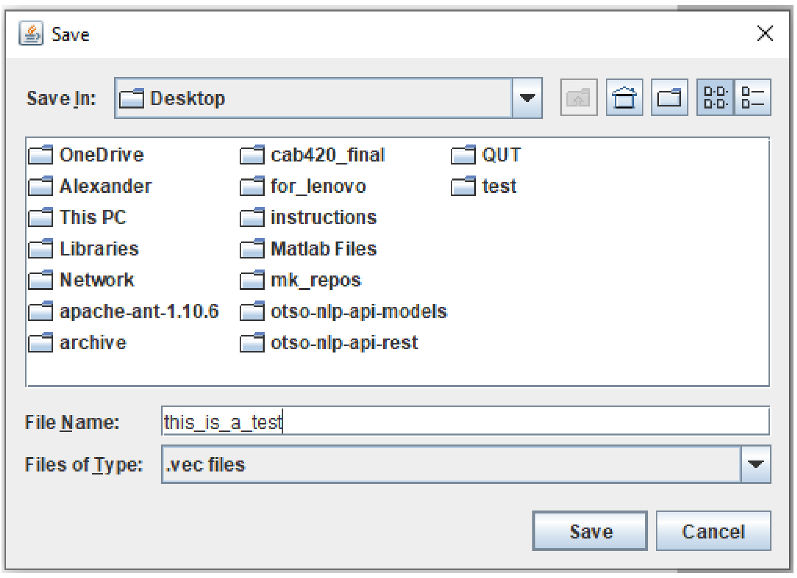
To **load, save** and **export** use the file menu.



Load will open the file chooser letting you navigate your system's file system while showing all files with the .vec (or .VEC) extension.



Save will open the file chooser letting you navigate to a directory to save the current state of the canvas in .vec (or .VEC) format. The file name can be entered with or without .vec (if without, it will be added automatically when saved).

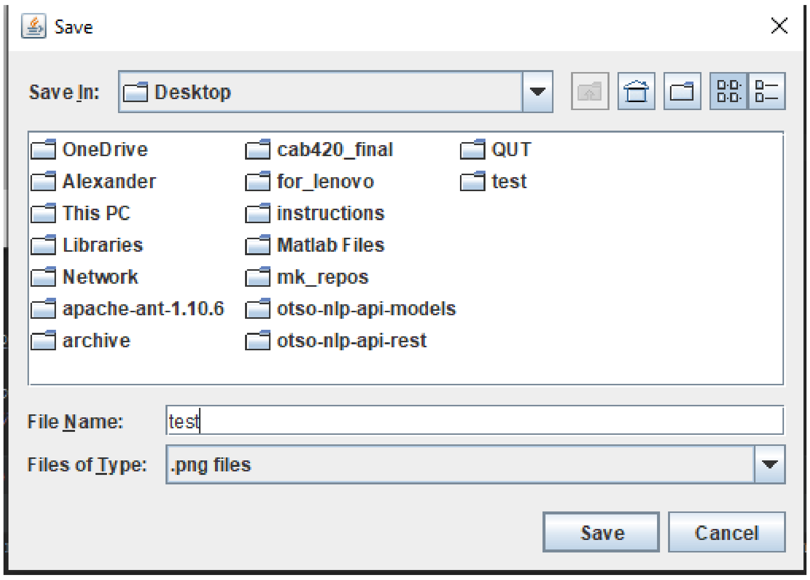


Save In: Desktop

File Name: this_is_a_test

Files of Type: .vec files

Export to .png will open the file chooser, letting you select the path and name of the file to export the current state of the canvas to **png**. Similarly to saving .vec, the filename can be entered with or without png (if without, it will be added automatically).



Save In: Desktop

File Name: test

Files of Type: .png files

Architecture

The program is split into two main packages, **graphicsManage** and **gui. graphicsManage** is where the shape classes are implemented, as well as file parsing for reading and writing .vec files. **gui** is where the graphical user interface is implemented. The following section outlines the classes used in each package and their purposes.

graphicsManage

DrawableVector

An interface containing abstract methods all the custom shape classes will need to implement. This includes the draw command, the get and set methods of the colors (pen and fill), the get function for command type (VectorCommand), and the *filled* method which returns a boolean determining whether a shape is filled.

The most prominent command that this interface requires is draw. This command takes a graphics object and two integers as parameters. These integers denote the size of the canvas, such that the points pertaining to each drawable vector (0.0 – 1.0 being the visible space) can be converted to absolute coordinates on the canvas when drawn. The program has been implemented such that the values representing the coordinates of the shape can be under 0 and over 1, this will just mean that points containing these values will be outside the visible space of the canvas.

FixedPointVector

This is an abstract class which extends drawable vector. It has been created for shapes which have the common property of having a fixed number of vertices. This is basically everything except for the polygon.

A FixedPointVector is a shape which is defined by two x points and two y points. This class therefore contains private values for x1, y1, x2, y2, which are stored as doubles (0.0 – 1.0 being visible drawing space on the canvas). This class also privately stores the Colour object for the pen.

A few methods are implemented, such as, the setting of the coordinates, the toString method which returns a string formatted with the numbers required in Ellipses, Rectangle and Line for .vec files (Point overrides this as it only has one x and y point) and the methods for the getting and setting of the pen colour.

Something to note is that the constructor and the setting of coordinates both validate x1, x2, y1 and y2 swapping the 2s with the 1s if x1 > x2 and y1 > y2 respectively. (this is due to the drawing of rectangles and ellipses needing to be from left to right, these are overridden with line however as it does not have this requirement.)

There are two versions of setCoordinates, one which accepts double values for x1, x2, y1, and y2 and one which accepts integer values. The integer overload also requires arguments for the canvas width and height such that the absolute coordinates on the canvas can be converted to doubles with 0.0 to 1.0 being the visible space for the x and y axis.

Ellipses

Ellipses extends FixedPointVector (which implements DrawableVector). It overrides the methods to draw and to return the command type. The constructor for this shape also takes parameters relating to its fill state, a boolean determining whether it should be filled and a color determining what the fill colour should be.

Rectangle
The rectangle extends FixedPointVector also and has very similar functionality to the Ellipses class. The main difference is that it draws a rectangle in the draw method.

Line
Line also extends FixedPointVector. However, in this case setCoordinates is overridden and the base constructor isn't used. This is because the draw method does not require x1, x2, y1, y2 to be swapped if x1 > x2 etc. Line will always return false for i*sFilled* and null for *getFillColour*.

Point
Point also extends FixedPointVector despite only having one x and one y value. In this case the super constructor is called with the same value for x1 and x2, and y1 and y2. Point will always return false for *isFilled* and null for *getFillColour*.

Polygon
The polygon class implements DrawableVector, however it does not extend FixedPointVector, as it requires an undetermined number of points to complete. It has methods which allows you to add new x and y points. With two overloads, one for doubles and one for integers (the one for integers also requires parameters for the canvas's width and height in order to be scaled). The draw method is also implemented to only draw lines up until the shape is completed (determined by a private boolean, which can be set to true with the method finishShape). Hence, there is a constructor which creates an unfinished polygon with no points, and one which creates a finished polygon with all points defined.

VecFileException
This is an exception class which is thrown in situations relating to file input and output. This is thrown in situations relating to incorrect file format and content, as well as incorrect arguments for commands.

VecFileManager
This is a class which contains only static methods for constructing .vec files from an ArrayList of DrawableVectors and generating an ArrayList of DrawableVectors based on a .vec file.

VectorCommand
This is an enum containing constants relating to the commands in .vec files. These commands are all related to the shape classes in this package, as well as commands for FILL (which can be followed with OFF, or a hex representation of RGB colour), and PEN (which will set the colour of the pen based on a hex representation of RGB colour).

gui

GUI
The Graphical User Interface class. This is implemented as a runnable so that it can be run on the event dispatch thread. This class glues all the gui components together while also containing a bit of logic to resize the canvas with a square aspect ratio when the window changes size. The static main method is in also in this class. This method will invoke the event dispatch thread with a new instance of GUI.

VecCanvas
This is the canvas class of the gui. It was implemented using a singleton design pattern, as many of the other gui classes share the same instance. The object contains an ArrayList of DrawableVectors (an interface which is described above.)

The class contains the paintComponent method, which will iterate through all the DrawableVectors in the list, calling their draw methods. There is also a method which will paint the graphics to a PNG file at a specified path. There is a method to undo the last instruction, which is bound to the keystroke of ctrl z.

The class contains a private field for currentInstruction which is a DrawableVector that hasn't been committed to the list yet. This is the instruction that can be manipulated with the mouse, which is achieved through adding an instance of CanvasMouse to the class's mouse listeners (described below).

CanvasMouse

This is a private class within the VecCanvas class. It extends MouseAdapter and overrides methods for mouse pressed, dragged and released. These methods will alter the current instruction's coordinates and add it to the list of instructions once it has been committed. As this is a private class within VecCanvas, it has access to all of its members.

VecColorButtonPanel

This is a class which extends JPanel to create the side panel of the gui with buttons allowing colour choosing interactions. The class implements ActionListener in order to listen to the two added buttons. The constructor for this class takes a VecCommandButtonPanel parameter, which is a panel of the gui that has buttons relating to drawing instructions. This parameter is passed so that the color panel can update which colors the command panel should use for the pen and fill when constructing a DrawableVector to send to the canvas.

VecCommandButtonPanel

This is also a class which extends JPanel. It contains the buttons which can be used to update the currentInstruction for the instance of VecCanvas. It also implements ActionListener, in order to determine which button was pressed, and update the VecCanvas's current instruction accordingly (i.e. which shape, and what parameters to use).

VecPainterMenuBar

This class extends JMenuBar. It is constructed to contain two JMenu's one for file and one for edit. File contains three items, one for save, one for load and one for exporting to csv. Edit contains two, one for Undo and one for Clear. All these items add an instance of MenuListener to their action listeners.

MenuListener

This is a private class within VecPainterMenuBar which implements ActionListener. This class contains private methods for the load file interaction, the save file interaction and the export file interaction (In a potential refactor, these functions could be a part of a public class in order to allow these methods to be more generally used i.e. if you wanted to trigger the save interaction with ctrl s.). The actionPerformed method determines which item was selected and calls the appropriate interaction (save, load etc) or canvas method (i.e. undo, clear).

VecFileChooser

This is a private class within VecPainterMenuBar which extends JFileChooser. This class overrides the accept and getDescription methods in order to be able to only show files of a certain type and base the description around that type. There is also a method to set the extension, in order to change this from .vec to .png for exporting.

Agile Methods

This program followed agile principles in its development. The following section outlines a couple of the principles that were used during the development. As there were no specific client interactions, and the group only consisted of one person a full agile approach wasn't exactly necessary or reasonable.

Deliver Early and Deliver Often

The software was updated frequently with new features. It was developed iteratively, admittedly much of the testing was completed after many of the features were developed. Additionally, much the GUI cannot be tested.

Working Software

It was ensured that the software that was committed was always functional, as to ensure the product on the repository was always in a working state. In the case anything was to go wrong, there would always be something to submit.

OOP

This section outlines how the four principles of object-oriented programming were followed in the development of this application.

Abstraction

Abstraction has been used throughout the assignment by using many classes to group underlying code and data. In these cases, the implementation of the classes' functionality is hidden. Some examples from the project are the shape classes. These all implement methods to draw and get information about the state. This allows external code to use these methods to fulfill tasks and retrieve relevant information, while not knowing exactly how this is done.

Inheritance

Inheritance has been used to create multiple classes of shapes which all inherit from the same interface. In addition, Rectangle, Ellipses, Line and Point, all share some similarly functionality so they all extend from the same abstract class FixedPointVector (this mitigates repeated code). Inheritance has also been used in the gui package, with extensions of JFrame, JPanel and JMenuBar, just to name a few. This allows base functionality to be inherited, with added extensions or changes to suit different requirements. It also means that these child classes satisfy the requirements of the parent class. This means that they can be used in situations where an instance of the parent class is required.

Encapsulation

Encaspulation has been used in many places throughout the assignment. In the gui package, VecCanvas has a private a member for the ArrayList of DrawableVectors. This can only be accessed and manipulated through public methods. The singleton design pattern is also achieved through encapsulation. The constructor of VecCanvas is private, while there is a static method to return the one instance of the canvas. This instance is also a private static member. These restraints mean that it can be ensured that only one instance of the VecCanvas can ever be created.

In the graphicsManage package, the Polygon has private members for its x points and y points (two ArrayLists). These lists can only be added to through the public method of *addCoordinates*. This ensures integrity of the lists as their lengths must be equal, only one x value and one y value can be added at a given time (unless the Polygon is initially constructed with all points defined).

Similarly, the x and y points for the FixedPointVector can only be set through public methods. This is because of the restraints mentioned earlier in the report for the drawing of rectangles and ellipses (i.e. needing to swap x1 and x2 if x1 > x2).

Polymorphism

Polymorphism is achieved in the program through inheritance, method overloading and method overriding. Method overloading can mainly be seen in the graphicsManage package, with all the shape classes having multiple constructors (one for a complete shape, and one for an incomplete shape). In FixedPointVector, there are two overloads for setting coordinates, one which takes the coordinates as integers along with the canvas width and height, and one which takes coordinates as doubles without the canvas dimensions. These result in the same data being stored in the instance, however the difference is that the method can now be used in two different contexts (when different data is available).

Another example is in VecCanvas, where there are two versions of paintComponent. One takes a graphics object by itself, and the other takes a graphics object along with a width and a height. In the second case, it is implied that this graphics object can belong to something other than the canvas (in the case of the program, this is the BufferedImage object – this is is used to save the state of the canvas to a png file.)

Polymorphism is also achieved through the method overriding we see with inheritance. In the case of classes which inherit from DrawableVector, the implemented methods can often be quite different. However, they all have the same interface which means that they can all be used in a context where a DrawableVector is expected. This allows the canvas to have a list of DrawableVectors, all with the same interface, but each with very different functionality (most noticeably the drawing functionality).