# 200312082

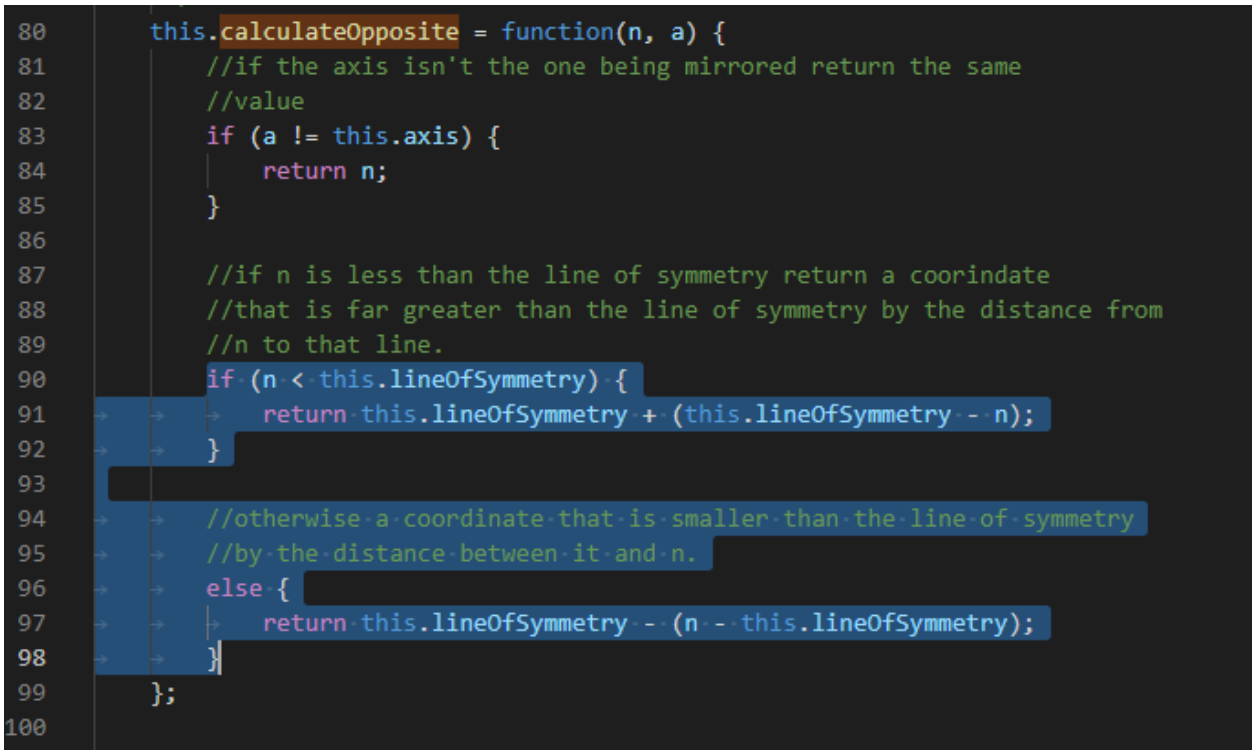
# Introduction

I have selected a drawing application from the “Introduction To Programming 2” module to analyse for module coupling and cohesion.
The program was designed for drawing on html5 canvas and several tools for drawing were developed in the program, these tools were organized into modules. The program was developed in javascript using p5js library.

# COUPLING

## 1. Control Coupling

```
18    this.draw = function() {
19        //display the last save state of pixels
20        updatePixels();
21        //do the drawing if the mouse is pressed
22        if (mouseIsPressed) {
23            //if the previous values are -1 set them to the current mouse location
24            //and mirrored positions
25            if (previousMouseX == -1) {
26                previousMouseX = mouseX;
27                previousMouseY = mouseY;
28                previousOppositeMouseX = this.calculateOpposite(mouseX, "x");
29                previousOppositeMouseY = this.calculateOpposite(mouseY, "y");
30            }
31            //if there are values in the previous locations
32            //draw a line between them and the current positions
33            else {
34                line(previousMouseX, previousMouseY, mouseX, mouseY);
35                previousMouseX = mouseX;
36                previousMouseY = mouseY;
37                //these are for the mirrored drawing the other side of the
38                //line of symmetry
39                var oX = this.calculateOpposite(mouseX, "x");
40                var oY = this.calculateOpposite(mouseY, "y");
41                line(previousOppositeMouseX, previousOppositeMouseY, oX, oY);
42                previousOppositeMouseX = oX;
43                previousOppositeMouseY = oY;
44            }
45        }
46        //if the mouse isn't pressed reset the previous values to -1
47        else {
```



```
80      this.calculateOpposite = function(n, a) {
81          //if the axis isn't the one being mirrored return the same
82          //value
83          if (a != this.axis) {
84              return n;
85          }
86
87          //if n is less than the line of symmetry return a coorindate
88          //that is far greater than the line of symmetry by the distance from
89          //n to that line.
90          if (n < this.lineOfSymmetry) {
91              return this.lineOfSymmetry + (this.lineOfSymmetry - n);
92          }
93
94          //otherwise a coordinate that is smaller than the line of symmetry
95          //by the distance between it and n.
96          else {
97              return this.lineOfSymmetry - (n - this.lineOfSymmetry);
98          }
99      };
100
```

***'mouseX'*** *and '***mouseY'*** are internal variables that get assigned as position of mouse changes by p5js library. *this.draw* function communicates with *this.calculateOpposite* function by passing the value of mouseX and mouseY.

The value of mouseX or mouseY is regarded as '***n'*** by *this.calculateOpposite* function and it's used as a flag for determining it's execution.

Likewise, depending on the value of '**a'** sent by *this.draw* function it affects the internal workings of *this.calculateOpposite* function.

Hence, this is a control coupling because *this.draw* function is determining the operation of *this.calculateOpposite* function by sending some flag.

## 2. COMMON ENVIRONMENT COUPLING

```
function mirrorDrawTool() {
    this.name = "mirrorDraw";
    this.icon = "assets/mirrorDraw.jpg";
    //which axis is being mirrored (x or y) x is default
    this.axis = "x";
    //line of symmetry is halfway across the screen
    this.lineOfSymmetry = width / 2;
    //this changes in the p5.dom click handler. So storing it as
    //a variable self now means we can still access this in the handler
    var self = this;
    //where was the mouse on the last time draw was called.
    //set it to -1 to begin with
    var previousMouseX = -1;
    var previousMouseY = -1;
    //mouse coordinates for the other side of the Line of symmetry.
    var previousOppositeMouseX = -1;
    var previousOppositeMouseY = -1;
```

```
    this.populateOptions = function() {
        select(".options").html(
            "<button id='directionButton'>Make Horizontal</button>");
        //  //click handler
        select("#directionButton").mouseClicked(function() {
            var button = select("#" + this.elt.id);
            if (self.axis == "x") {
                self.axis = "y";
                self.lineOfSymmetry = height / 2;
                button.html('Make Vertical');
            } else {
                self.axis = "x";
                self.lineOfSymmetry = width / 2;
                button.html('Make Horizontal');
            }
        });
    };
```

```
80      this.calculateOpposite = function(n, a) {
81          //if the axis isn't the one being mirrored return the same
82          //value
83          if (a != this.axis) {
84              return n;
85          }
86
87          //if n is less than the line of symmetry return a coorindate
88          //that is far greater than the line of symmetry by the distance from
89          //n to that line.
90          if (n < this.lineOfSymmetry) {
91              return this.lineOfSymmetry + (this.lineOfSymmetry - n);
92          }
93
94          //otherwise a coordinate that is smaller than the line of symmetry
95          //by the distance between it and n.
96          else {
97              return this.lineOfSymmetry - (n - this.lineOfSymmetry);
98          }
99      };
100
```

***this.axis*** was defined at the beginning of the main module (mirrorDrawTool). *this.populateOptions* is modifying the axis value through *self.axis* while *this.calculateOpposite* is reading the value of *this.axis* for its internal execution.

Likewise, ***this.lineOfSymmetry*** was defined at the top level *this.populateOptions* is changing its value based on value of width or height. *this.calculateOpposite* also reads this value by comparing with ***n*** and then modifies the value afterwards.

Therefore, this is a common-environment coupling because both functions access some data area defined globally.

# COHESION

## 1. Sequential Cohesion

```
33    this.populateOptions = () => {
34      input = createFileInput(handleFile);
35      input.parent(Gopt);
36    };
```

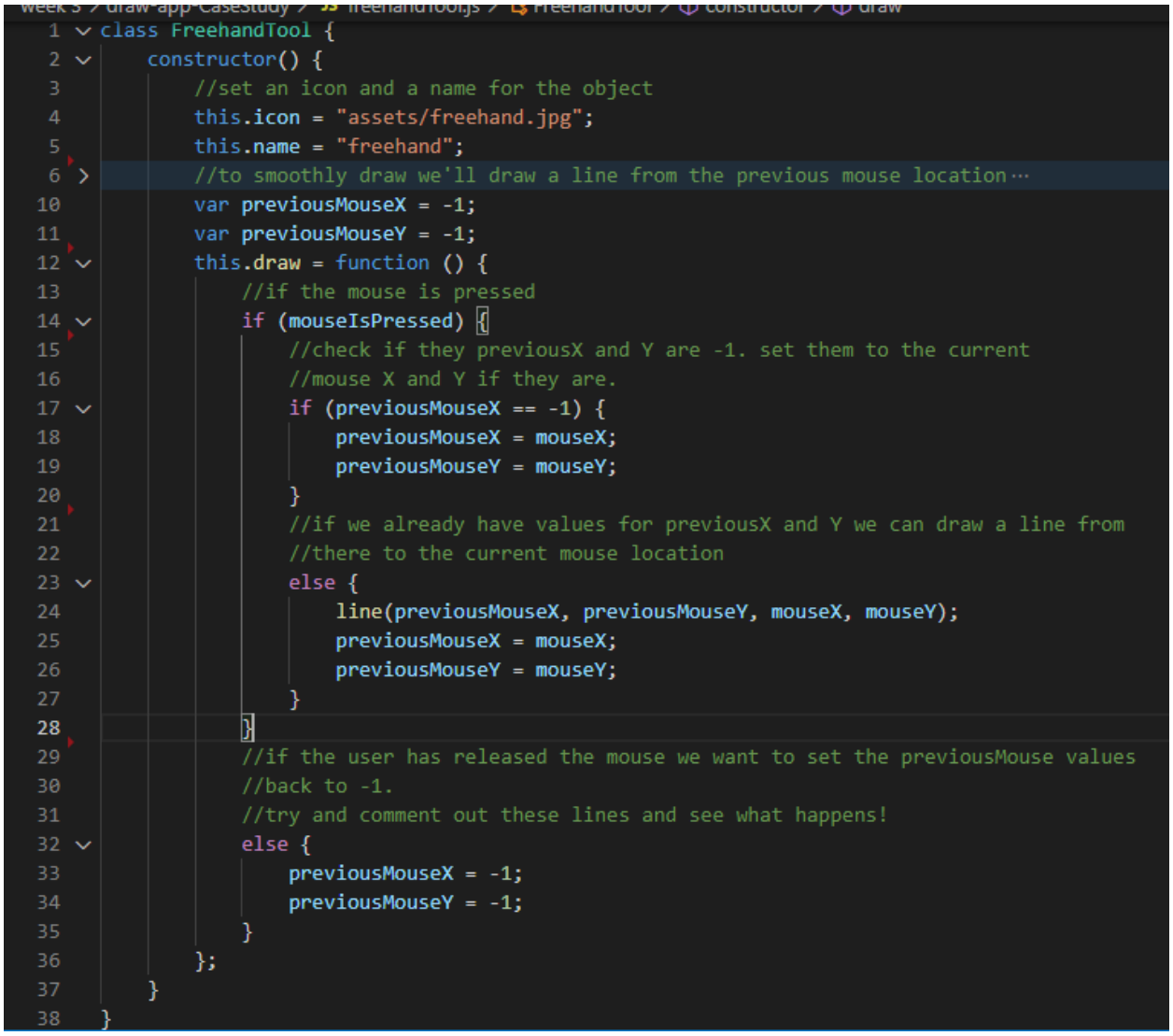
```
12    const handleFile = (file) => {
13      if (file.type === "image") {
14        createImg(file.data, "", "", (image) => {
15          img = image;
16          if (!slider) {
17            slider = createSlider(
18              5,
19              img.size().width * 1.5,
20              img.size().width / 2,
21              5
22            );
23            slider.parent(Gopt);
24            img.hide();
25          } else {
26            img.hide();
27          }
28        });
29      } else {
30        img = null;
31      }
32    };
```

The *handleFile* function gets called when the *createFileInput* assigned to input has been created in the DOM. This *handleFile* function takes in the file and it then calls another function named *createImg* (internal function in p5js) passing the data of the file obtained. The *createImg* function itself has a callback which passes the *p5.Element* (named image in the code above) data returned to the callback function, it doesn't stop there.

The callback to *createImg* also call an internal p5js function - *createSlider*, which use the return value of *createImg* for its execution.

This is a **sequential cohesion** because the file data processed by *handleFile* is passed as an input to createImg and the returned value of *createImg* serves as an input to its success call back function, again the callback function serve the input of *createSlider.* This is the main reason why they are all in the same module.

## 2. Functional Cohesion



```
class FreehandTool {
    constructor() {
        //set an icon and a name for the object
        this.icon = "assets/freehand.jpg";
        this.name = "freehand";
        //to smoothly draw we'll draw a line from the previous mouse location…
        var previousMouseX = -1;
        var previousMouseY = -1;
        this.draw = function () {
            //if the mouse is pressed
            if (mouseIsPressed) {
                //check if they previousX and Y are -1. set them to the current
                //mouse X and Y if they are.
                if (previousMouseX == -1) {
                    previousMouseX = mouseX;
                    previousMouseY = mouseY;
                }
                //if we already have values for previousX and Y we can draw a line from
                //there to the current mouse location
                else {
                    line(previousMouseX, previousMouseY, mouseX, mouseY);
                    previousMouseX = mouseX;
                    previousMouseY = mouseY;
                }
            }
            //if the user has released the mouse we want to set the previousMouse values
            //back to -1.
            //try and comment out these lines and see what happens!
            else {
                previousMouseX = -1;
                previousMouseY = -1;
            }
        };
    }
}
```

This is a functional cohesion because the whole module works in order to achieve a singular goal: Drawing a regular/irregular line on the canvas of the application.

This module makes use of the pre-defined variable in p5js library (*mouseIsPressed*, *mouseX, mouseY* and the *line* function). It makes no call to external modules and it's data is only used by itself (except for icon and name, which is required for all tools) hence the use of var in it's variable declaration.

This module has no complexity, the goal is clearly spelt out.