Project 4 - Shape Paint v2.0

CSC-121, Fall 2025

Overview

To practice the principles of object oriented programming, arrays, and lists, we are going to build a drawing program. Our drawing program will allow the user to create images by placing shapes onto a canvas. Users will be able to change the type of shape (Circle, square, rectangle, or triangle), the fill and stroke/outline colors of the shapes, and the size of the shapes. When completed, users can save their canvas as an image file.

Program Features

Shape Types

Our program supports drawing four different shapes: Circles, Squares, Rectangles, and Triangles. Shapes can be changed with the 1 key on the keyboard, and placed on the canvas with a left mouse button click.

Colors

Each shape can change its stroke/outline color, and its fill color. Stroke color can be changed by pressing 2 and fill color changed by pressing 3.

Size

The size of each shape can be changed by pressing the + and - buttons.

Undo

If a user makes a mistake, they can undo placing the last shape by clicking the right mouse button or the Backspace button. Users are able to undo not just the last shape, but all previous shapes placed. In addition, users can also start their drawing over from scratch by pressing the DEL key.

Save

Finally, users can save out their finished drawing to an image file by pressing the 0 key. Each time 0 is pressed it will save the image out as a unique filename.

Tools

Our ShapeDraw program uses the Processing real-time interactive graphics library to create the window and draw elements onto the screen. All of the code to setup the window through Processing has been written for you. Also, all of the code to receive user interaction such as mouse clicks and keyboard commands have also been written for you in advance.

You will need to know a few things about Processing in order to write your code. First, all files that use Processing commands must import the Processing library. You can do this by including the line *`import processing.core.*;`* at the top of any Java class files you create that use Processing commands or features.

The class that maintains the window and creates the drawing canvas is called a "PApplet", or Processing Applet. Our ShapePaint class is a type of PApplet. To draw your shapes, or to set the stroke or fill color, you will need a reference to the ShapePaint class and call the appropriate methods.

There are Processing commands to draw each of the shapes you are asked to draw for this project, and there are dedicated pages in the documentation for each. For example, the documentation page for drawing circles can be found here: [circle() / Reference / Processing.org](). There is also an example of drawing a circle in the ShapePaint class, which is used to draw a dot at the mouse cursor. You can find the full list of Processing commands at [Reference / Processing.org]().

Shape Classes

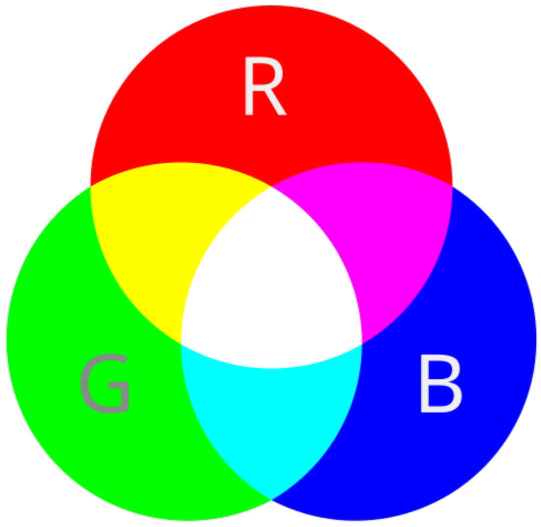
There are four shapes our program supports: Circles, Squares, Rectangles, and Triangles. Each of the four shapes will be represented by one of the following three classes.

Circle	Rectangle	Triangle
- x : float - y : float - diameter : float - colorStroke : int - colorFill : int	- x : float - y : float - width: float - height: float - colorStroke : int - colorFill : int	- x : float - y : float - diameter : float - colorStroke : int - colorFill : int
+ Circle(inX : float, inY : float, d : float, colorS : int, colorF : int) + draw(canvas : PApplet) : void	+ Rectangle(inX: float, inY : float, inSideLength: float, colorS : int, colorF : int) + Rectangle(inX: float, inY : float, w: float, h: float, colorS: int, colorF: int) + draw(canvas : PApplet) : void	+ Triangle(inX : float, inY : float, inSideLength: float, colorS: int, colorF: int) + draw(canvas : PApplet) : void

Each shape has a set of fields for the canvas position as a decimal value X, Y pair, int fields for the stroke (outline) and fill colors, and one or two decimal fields to represent the size of the object as either diameter or side lengths. Each class also has a constructor that takes in initialization values for each matching field. The Rectangle class has two constructors: One that takes in two side lengths (Width and height), and one constructor that takes in only one side length (For creating a square). Each class also has a draw method, which will contain unique code to draw each shape instance onto the canvas.

Colors

Our drawing program will allow users to change the color of shape outlines, shape fills, and the background of the canvas. Colors are represented as having four individual components: Red, Green, Blue, and Alpha (Transparency). By changing the value of each component, we can create different colors. The RGB Color Model follows the rules of "additive colors". You can read more about RGB and additive colors at: The ultimate guide to RGB | Oppaca.



Colors are stored as integers. To create a new color, we call the PApplet color methods (Reference page: color() / Reference / Processing.org). There are a few different color method headers, so let's look at just one of them:

Syntax
`color(v1, v2, v3, alpha)`

Parameters
`v1(int, float)`: Red or hue values relative to the current color range
`v2(int, float)`: Green or saturation values relative to the current color range
`v3(int, float)`: Blue or brightness values relative to the current color range
`alpha(int, float)`: Transparency relative to current color range

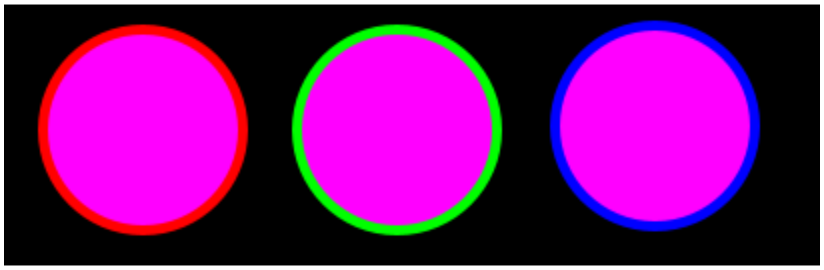
Return
int

This tells us that this color method is called with four int or float parameters, and the return type is an int. The input parameters in order represent the amount of red, green, blue, and alpha in the color. Also, the descriptive text on the documentation tells us that each component is in the range of 0 to 255. So, if we wanted to create a color that was pure red and fully opaque, we would set the red and alpha values to 255, and the green and blue values to 0. To create a color variable for red, we could write something like the following:

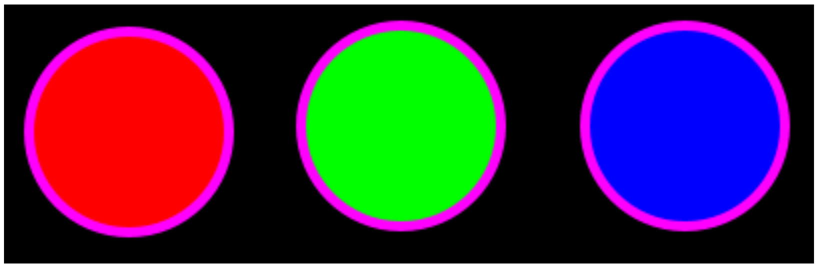
```
final int RED = color(255, 0, 0, 255);
```

After creating a color, we can set the stroke and fill colors of our shape by calling the PApplet stroke and fill methods, respectively.

Changing the stroke color: stroke() / Reference / Processing.org



Changing the fill color: fill() / Reference / Processing.org



PApplet Canvas

The ShapePaint class is derived from the PApplet class. Thus, if you want to draw onto the canvas from your shape classes, you will need to take in a reference to the ShapePaint instance as input argument to your shape's draw method.

Another thing to note is that the position of an object within the PApplet canvas is determined by an x and y position. The top left of the canvas is the origin of the coordinate system and represents x=0 and y=0. The x coordinate increases as we go from the left to the right of the canvas, and the y coordinate increases as we go from the top to the bottom of the canvas. bottom right of the canvas. The coordinates at the bottom-right corner of the canvas for a 500x500 pixel size canvas would be x=499 and y=499.

Student Requirements and Tasks

Students must download the project framework and import it into Eclipse as a starting point. The ShapePaint class has a few methods already written for creating the window/canvas, cycling between the different shape drawing modes, and accepting input from the mouse and keyboard. Students are expected to write the different shape classes, create and maintain a list of shape instances, manage an undo list, and create an array of colors. Students may do these tasks in any order they choose.

☐ Write Shape Classes

The three shape classes are defined elsewhere in this document with accompanying UML diagrams. Each shape has at least one constructor, and a draw method.

☐ Create Shape Instances

When the user clicks the left mouse button, an instance of the shape matching the current mode should be created.

☐ Shape Lists

Shape instances should be added to a list of shapes of that type. I.E. There should be a list of circles, a list of rectangles, and a list of triangles. When the user creates a circle, an instance of the circle class with the properties the user specified (Position, size, stroke color, and fill color) should be added to the circle instance list.

☐ Draw Shape Instances from Lists

When the ShapePaint's "draw()" method is called, we should iterate through each of our shape lists and call the draw() method of each shape instance.
NOTE: The order the shapes are drawn in does not matter and will be dependent on which shape list you draw first. Since we will be drawing each list one at a time, some shape types will always be in front of or behind certain other shape types.

☐ Draw Shape at Mouse Cursor

We should create and draw a temporary shape at the mouse cursor's position to show what our image would look like if we were to add the current shape to the canvas.

☐ Create Array of Colors

Create an array of color values. The colorIndexStroke, colorIndexFill, and colorIndexBackground values should properly cycle through the possible indices of the list when the user taps the 2, 3, or 4 buttons on the keyboard. When you create a shape instance, you should pass along the currently selected stroke and fill colors to the constructor of that shape. Also, you should properly set the background color based on the colorIndexBackground value.

A few color values have been created for you in the sample code. You should make more colors, and then add them to your color array.

☐ Clear Canvas

The clearCanvas() method should completely clear all of your shape instance lists to clear the canvas.

☐ Undo

Users should be able to undo placing a shape. This action should remove the last shape instance added to a particular shape list. It is up to you to decide how to implement this. One potential solution could be to create a list of shape modes so you know which order shapes have been added in.

☐ Create a Masterpiece

Once you have completed all of the above steps, create a drawing with your program. Save a screenshot with the ‘0’ key on your keyboard and submit the image along with your code.

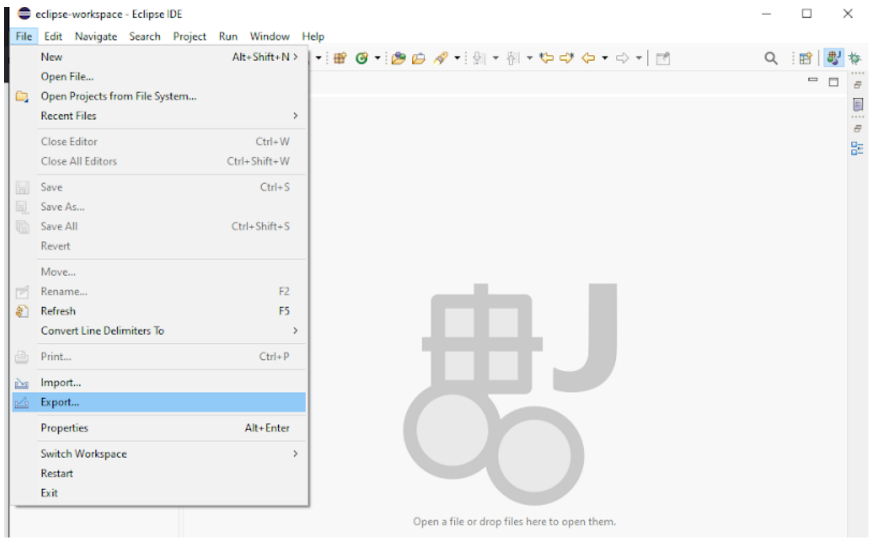
☐ Create an Executable JAR File

In addition to your code, manual, video, and masterpiece, export your project as an executable JAR file from eclipse. Ensure that you can run the JAR on your own local machine after you create it. Submit your JAR file along with your other project files.

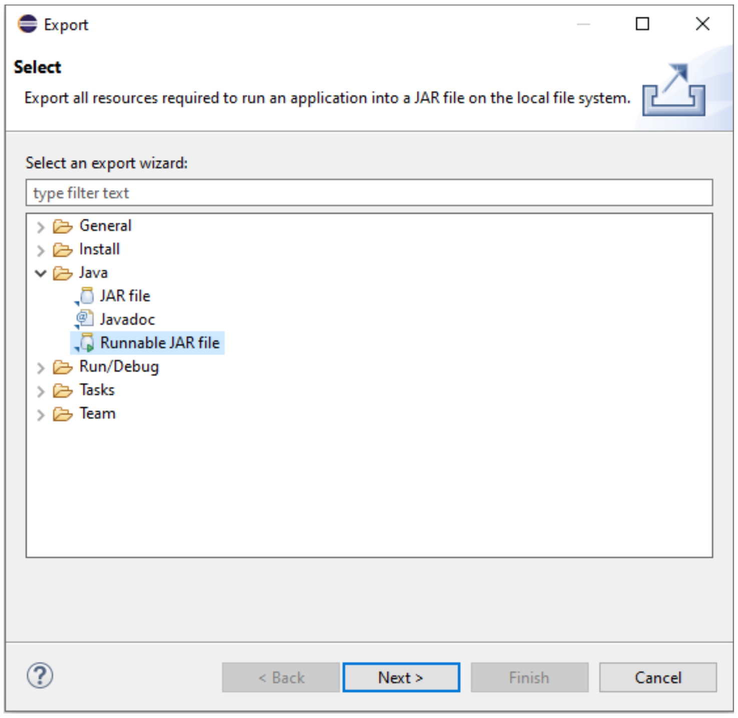
Exporting an Executable JAR File

It's nice that we can run our code in Eclipse, but it would be nice if we could package up our program and give it to other people. Java lets us do that by bundling our bytecode up into a JAR file. Anyone with a Java runtime environment can run our program from the JAR.

1. In Eclipse, select "File -> Export".



2. For the export type, select "Java -> Runnable JAR file".

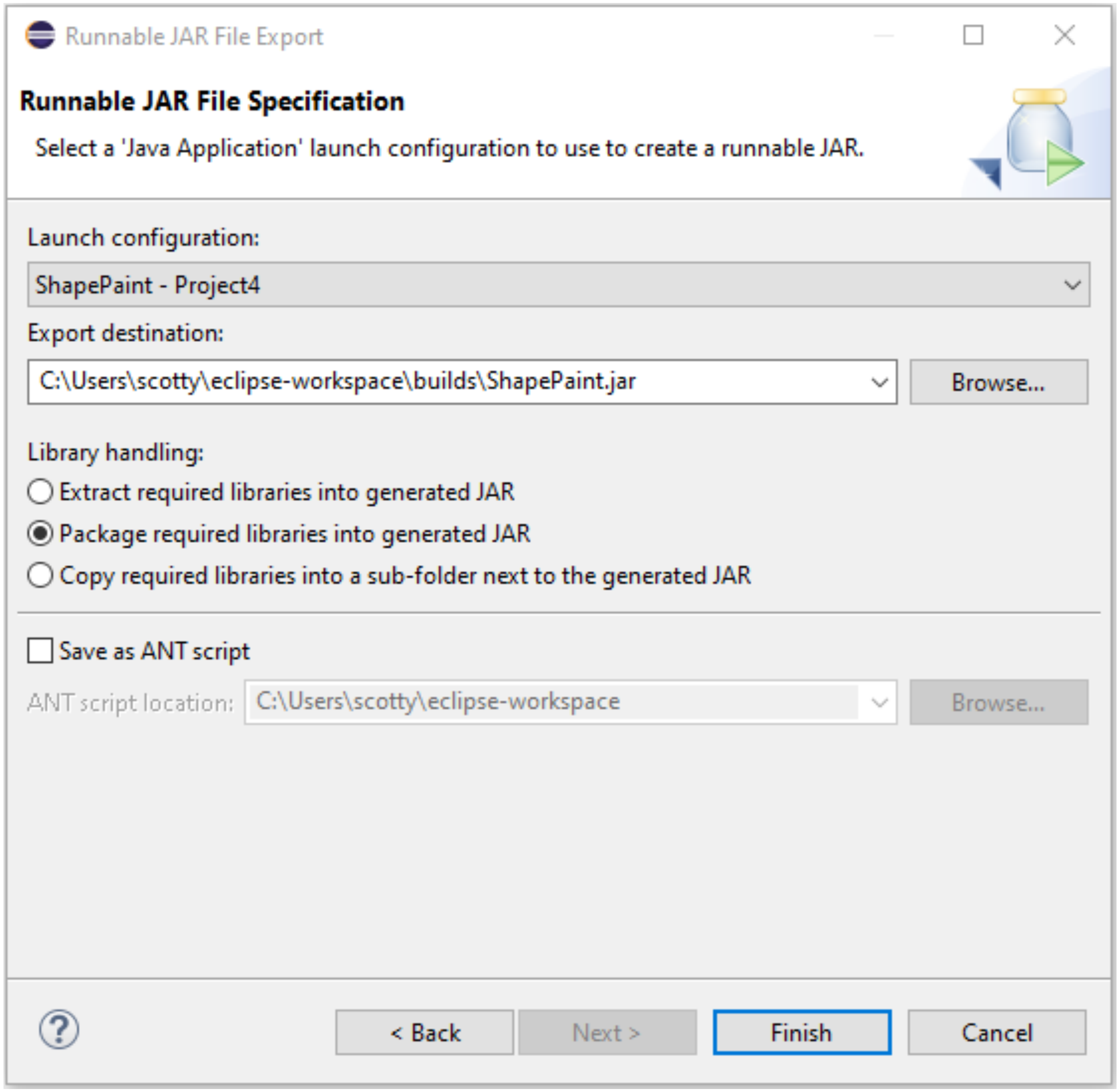


Select

Export all resources required to run an application into a JAR file on the local file system.

Select an export wizard:

3. For the “Launch configuration”, select your project and the class that contains your "public static void main" method (For this project, it should be something like “ShapePaint - Project4”). Choose a location to export the JAR file to on the “Export destination” box. Under “Library handling”, click “Package required libraries into generated JAR”. Then click Finish.


Runnable JAR File Export

Runnable JAR File Specification

Select a 'Java Application' launch configuration to use to create a runnable JAR.

Launch configuration: ShapePaint - Project4

Export destination: C:\Users\scotty\eclipse-workspace\builds\ShapePaint.jar

Library handling:
- ○ Extract required libraries into generated JAR
- ● Package required libraries into generated JAR
- ○ Copy required libraries into a sub-folder next to the generated JAR

☐ Save as ANT script

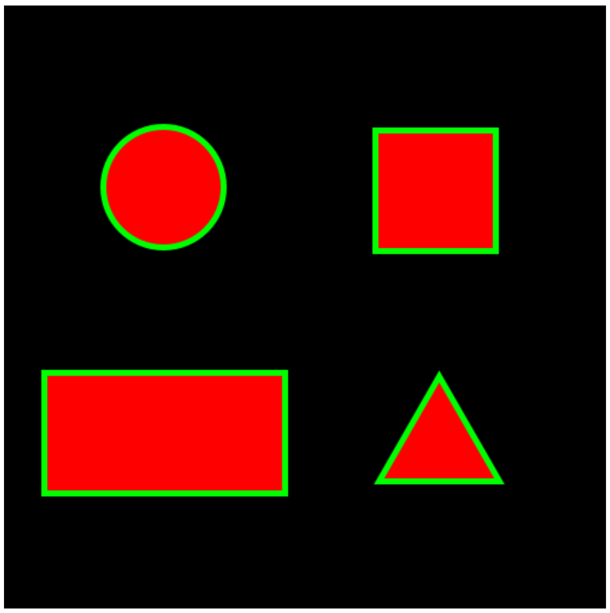
ANT script location: C:\Users\scotty\eclipse-workspace

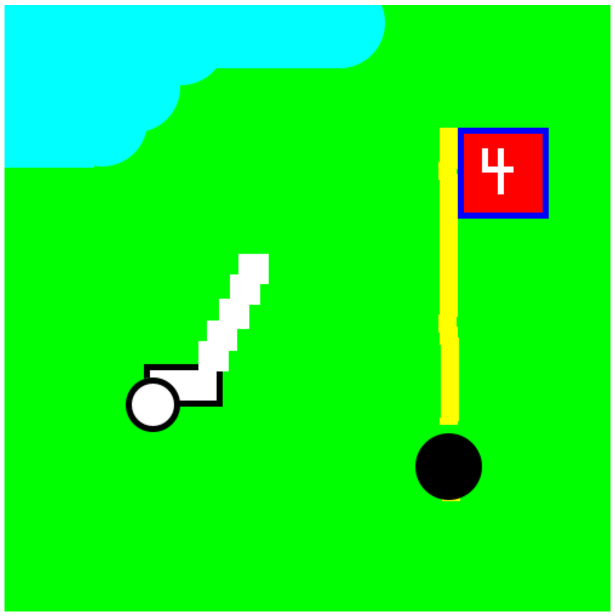
4. Test your JAR file by double-clicking it and see if your program launches. Depending on your operating system, you may need to give the JAR file executable privileges before it will run.

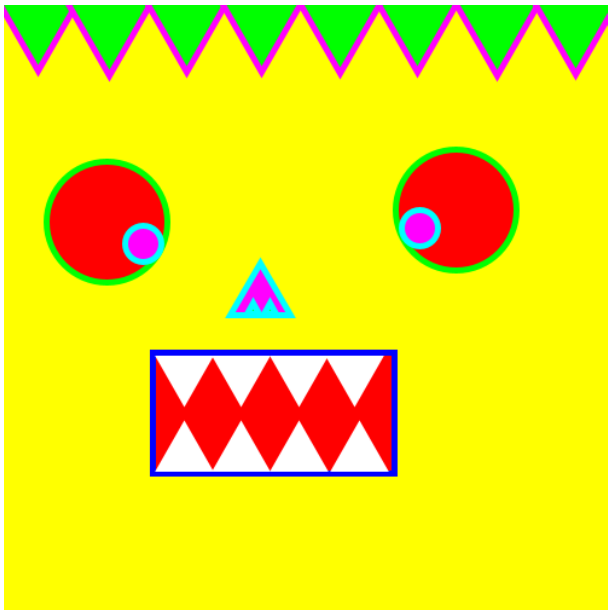
Extra Credit - Art Contest!

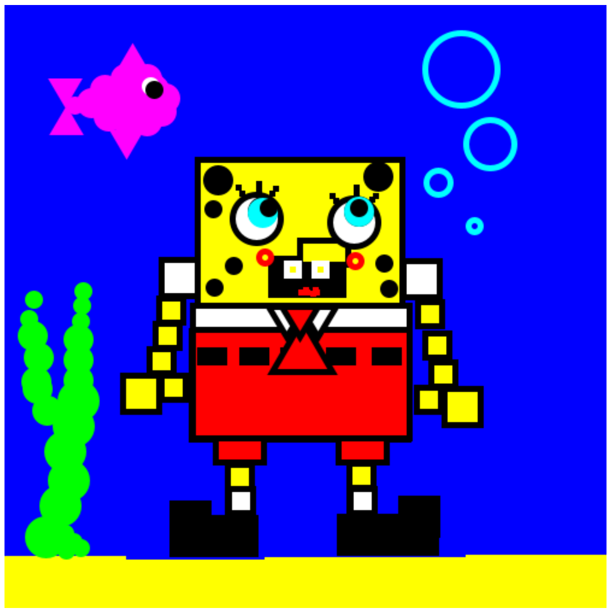
Be sure to include at least one drawing you created with your completed ShapePaint program as part of your project submission. The TFs will act as judges of an in-class art contest where the winners and runner ups can receive extra credit points. Don't hold back - embrace your inner digital artistry!

Below are some examples of images you can make with our ShapePaint program. What will you create?









Notes on genAI usage:

You are encouraged to use an AI tool while writing your code. For example, you may want to ask it to explain a concept related to your program, such as what operator to use to check if two values are equal. Or, if you find you have a syntax error, you could leverage genAI as part of your debugging process. However, you should not trust AI tools blindly, or copy and paste AI-generated code without understanding how it works. ***You should not use concepts that we have not yet covered in class, even if AI suggests them***. Keep in mind that you will need to explain your code on video, and must understand your code to do so.

Explain your work

Manual

In addition to your project code .java files and video, you must write a short manual to accompany your project. Name your plain text manual "README.txt". Include any information you deem necessary to running and using the program.

Video

A final part of the project is to explain your work and demonstrate your understanding of the program you created.

You will share your explanation in a 5-minute video after creating a detailed outline of the points that you want to make. Specifically, you must explain the following:

- The intended functionality of this program.
- Your process and the steps you took to design and develop this program.
- How the code executes in one of your most complex sections.
- The lessons you learned through this project

Each part should be at least 1-minute long. You can use a standard cellphone or webcam to record your video. If you use a screen recording program, we still need to be able to hear your voice explaining the code and project.

Please submit your video using a standard video format, like .mp4 format. If you are unable to submit the video file along with your submission on canvas for any reason, you may instead upload the video to a free service such as YouTube or Vimeo and include a URL link to the video with your project submission in a plain text file named "video.txt". Your video must remain accessible to grading staff until grading is complete.

Grading - Deliverables and Evaluation Criteria

You will be responsible for the following deliverables. The weighting of each component is indicated below. It's important that you refer to the Project 1 Rubric for detailed information about how each deliverable will be evaluated.

Checkpoint (10%)

Each student must meet with the professor or a TF in an in-person meeting before the project deadline to discuss their project, and any problems they may be encountering. Feedback will be offered to help students with building their project. Be sure to ask the professor/TF to count your project check-in.

Video Explanation (20%)

You must submit your video (Or a plain text file with a link to your video) containing your explanation along with your detailed outline by the due date.

Code submission (60%)

You must submit your code by the due date. *Please ensure it is possible to successfully compile and run your Java program!* If any additional instructions are required to run your project, please include them in your manual.

Additional Documents (10%)

- Manual
  You must submit a plain text manual file explaining how to use your program. This should cover an overview of what the program is, and any special instructions for running or using the program (Such as keyboard controls).

- Executable JAR
  Export your project as an executable JAR file from eclipse. Instructions for creating an executable JAR are included in this document. Ensure that you can run the JAR on your own machine after you create it. Submit your JAR file along with your other project files.

- Masterpiece
  Create a drawing with your program. Save a screenshot with the ‘0’ key on your keyboard and submit the image along with your code.

Late Policy

Projects submitted after the deadline will be subject to a 20% late penalty.