CS102 – Algorithms and Programming II
Programming Assignment 5
Fall 2023

ATTENTION:

- Compress all of the Java program source files (.java) files into a single zip file.
- The name of the zip file should follow the below convention:
  CS102_Sec1_Asgn5_YourSurname_YourName.zip
- Replace the variables “Sec1”, “YourSurname” and “YourName” with your actual section, surname, and name.
- You may ask questions on Moodle and during your section’s lab.
- Upload the above zip file to Moodle by the deadline (if not, significant points will be taken off). You will get a chance to update and improve your solution by consulting with the TAs and tutors during your section’s lab.

GRADING WARNING:

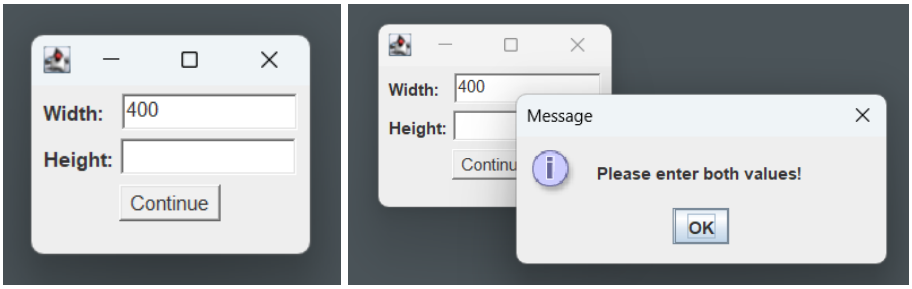
- Please read the grading criteria provided on Moodle. The work must be done individually. Code sharing is strictly forbidden. We are using sophisticated tools to check the code similarities. The Honor Code specifies what you can and cannot do. Breaking the rules will result in disciplinary action.

Simple Paint With Recursive Laser Fill

For this assignment, you will implement a simple paint application where we can draw on canvas using different pen sizes and colors. You will also implement a recursive laser fill tool that works similarly to the paint bucket in painting applications but only fills along a vertical line. The application will consist of three different frames that interact with each other; consequently, we expect you to separate view and control classes.

Implement a Controller class that includes the objects of SettingsFrame, ToolsFrame, and PaintFrame. These three classes, which all extend JFrame, should include a reference to the Controller class so that they can call its methods. Any input received from SettingsFrame, ToolsFrame, or PaintFrame classes should be directed to the Controller class so that it makes the necessary changes.

SettingsFrame is the welcoming screen of the application. The user should enter the desired width and height for the painting canvas. Make sure the user cannot enter invalid values into the corresponding TextFields. After the user clicks the continue button, check if the given inputs are reasonable (i.e., 0 < width < 1000 and width contains only numbers), and show an error message if not.



You may use `JOptionPane.showMessageDialog(null, "Error Message!");` method for displaying such error messages. Given the inputs are correct, clicking on the continue button should inform the Controller class so that it can hide SettingsFrame, create a new BufferedImage to act as our painting canvas, send this image to the PaintFrame for displaying, and show PaintFrame and Tools Frame on screen.

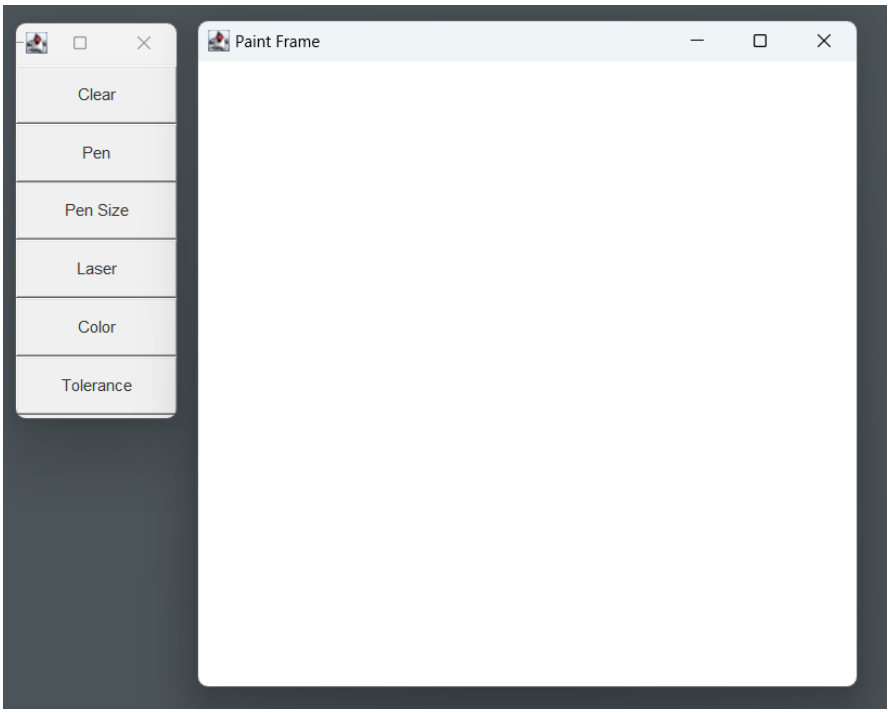
The image we paint should be stored as a BufferedImage in the Controller class. Since we will display this image in PaintFrame, a reference to this BufferedImage should also be included in the PaintFrame class. You can construct the image using the following code:

```
new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
```

BufferedImage can be drawn on the screen using the drawImage method of Graphics. Suppose we are in the PaintFrame class that extends JFrame and we have the BufferedImage in the image variable; you can override the paint method of JFrame to draw the image on screen using the following code:

```
public void paint(Graphics g) {
   g.drawImage(image, 0, 0, null);
}
```

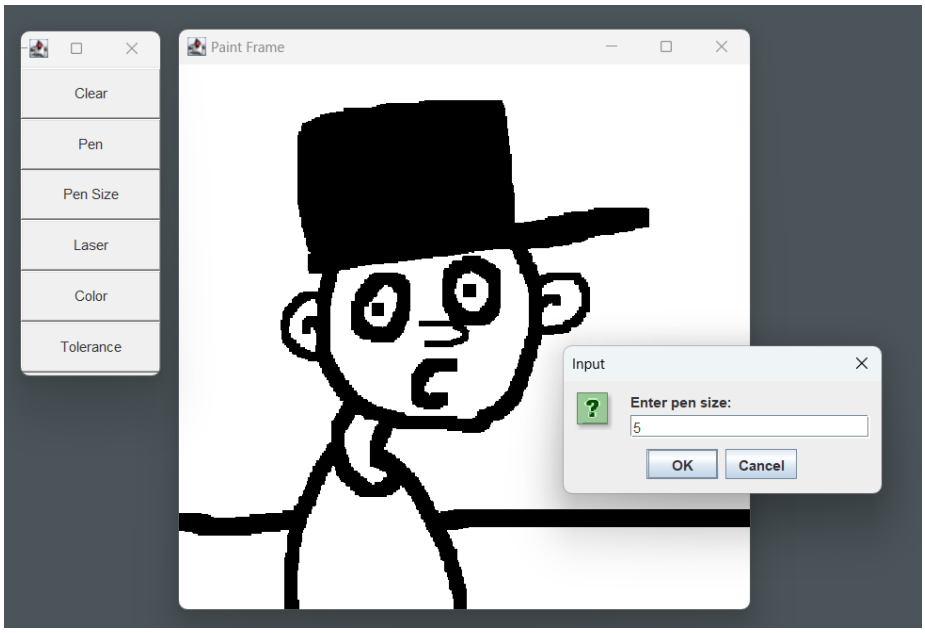
PaintFrame and ToolsFrame will appear side by side so that user can select the tool to use from the ToolsFrame and use it on PaintFrame:



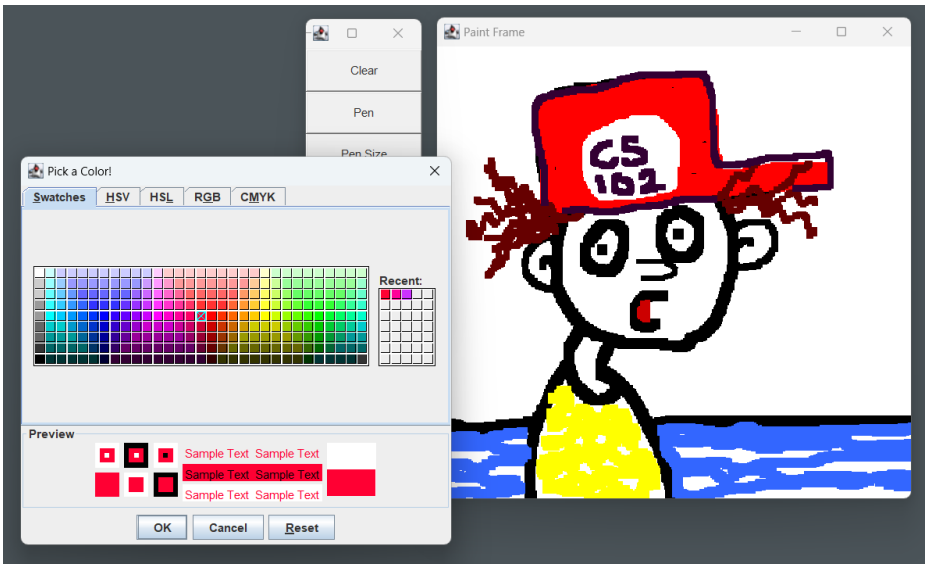
PaintFrame should implement MouseInputListener interface so that it can detect mouse input and position. This class should direct the mouse input to the Controller class to operate according to the current tool. The Controller class should keep track of which tool is currently active. The pen tool should be active at the beginning.

ToolsFrame includes a set of buttons using the GridLayout. This class should direct any button triggers to the corresponding methods in the Controller class. The **Clear** button should clear the current painting canvas, meaning it will set all of its pixels to white. The **Pen** button enables drawing with a pen on PaintingFrame. We move the mouse on PaintingFrame while pressing the mouse button, which paints a set of pixels on our BufferedImage. Instead of painting one pixel at a time, our mouse input should paint a square area with a side length of PenSize * 2 + 1. That is, we paint the pixel that our mouse is on and the pixels that are PenSize away in each direction. You may use the `setRGB(int x, int y, int color)` method of BufferedImage to color a pixel, do not forget to repaint your PaintFrame after coloring the pixels so that you can see the changes on screen.

The **Pen Size** button opens up a dialog where the user can enter the desired Pen Size. This enables drawing using different sizes with the Pen tool. You may use `JOptionPane.showInputDialog("Enter pen size:","");` method for showing such dialog and receiving the user input as a string. Make sure the input is valid before converting it to an integer and passing it to the Controller class.



The **Color** tool opens a color chooser (you may use `JColorChooser.showDialog(null, "Pick a Color!", Color.WHITE);` method) to pick a color to use with pen and laser tools.



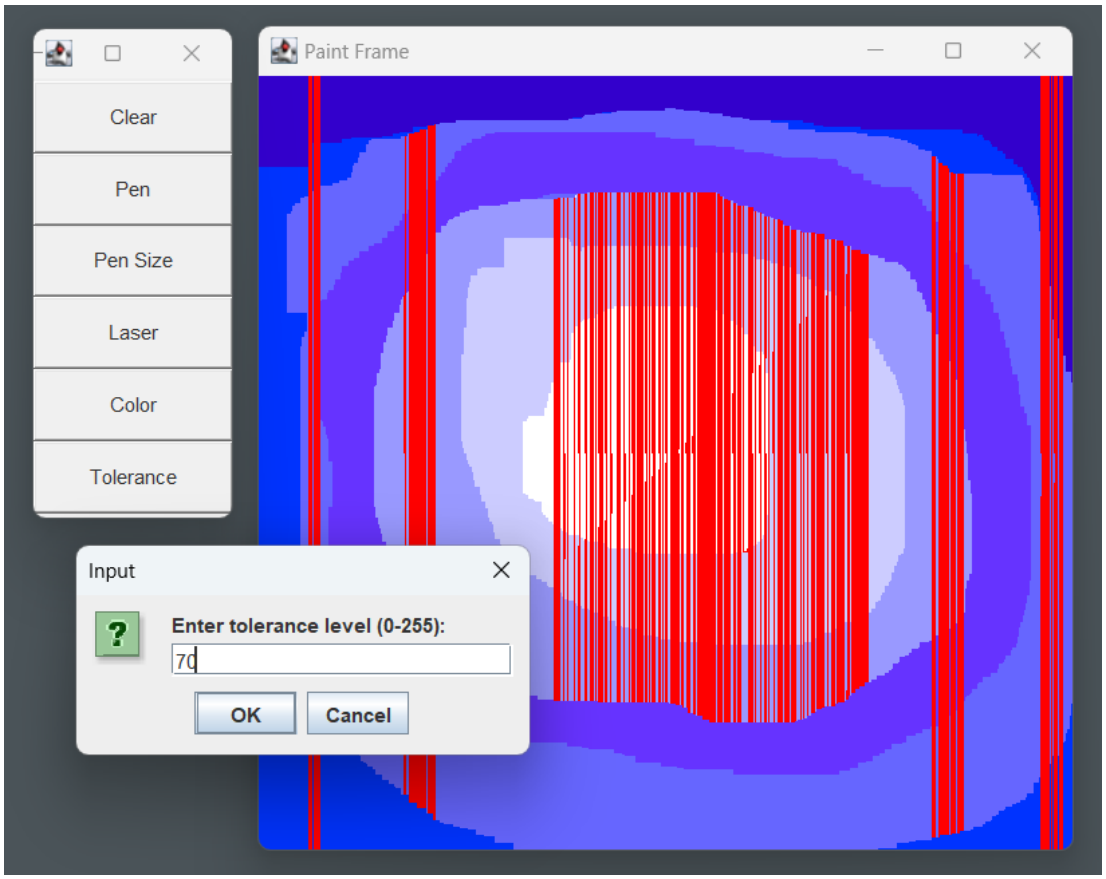
The **Laser** tool fills the pixels along a vertical column. From the mouse location, we go both up and down, painting the pixels as long as the color of the pixels that we paint is similar to the pixel that we start. Suppose we start on a white pixel; we will paint all the neighboring white pixels until we encounter a different color. You should implement this method using recursion. This tool will enable us to draw vertical lines cut at the contours of the drawing. You may check the attached “Laser.gif” to see how this tool behaves.

The recursive method needs the x and y coordinates of the pixel, the color of the pixel where we started, and the color to paint this pixel. Make sure you check the boundaries of the image before trying to get or set any pixels. You will also need to keep a boolean array to track which pixels are visited before; otherwise, you will go into infinite recursion, causing a stack overflow (Alternatively, you can have separate recursive methods that go only up and only down). Recursion returns if current coordinates are out of bounds, the current pixel's color is not similar to the pixel we started originally, or the pixel was visited before. Otherwise, paint the current pixel to the target color, set the current pixel as visited, and continue recursion for the pixel one above and one below.

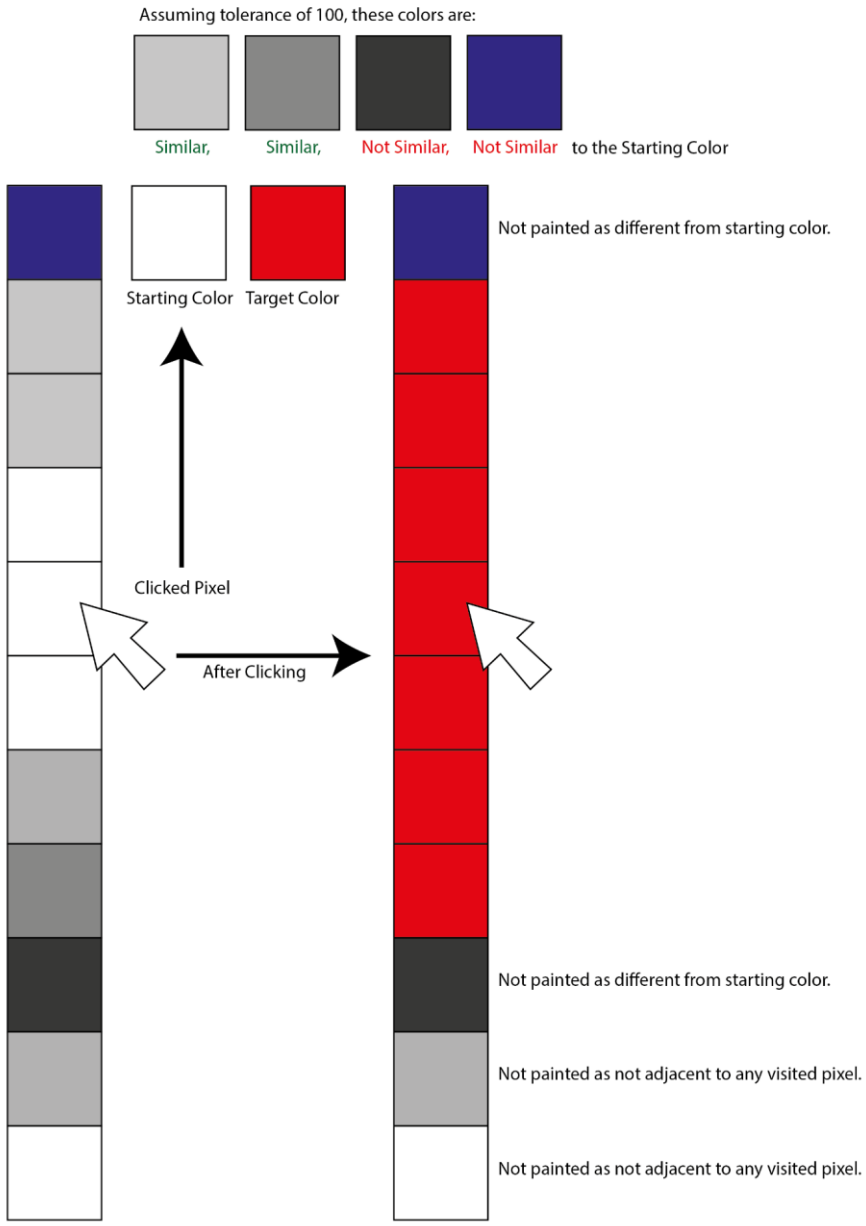
At the beginning, you may consider the underlined similarity as being the same color; however, we will assume the colors are the same within a tolerance margin. Suppose we have two Color objects a and b; these colors are similar if:

|Red(a)-Red(b)| + |Green(a)-Green(b)| + |Blue(a)-Blue(b)| / 3 < Tolerance

Where Tolerance is an integer in the range [0,255] kept in the Controller class; and Red(), Green() and Blue() refer to the red, green and blue amount of the color which can be received using `getRed(), getGreen(), and getBlue()` methods. In short, we want the average color difference of the two colors to be in a distance determined by the tolerance variable. The default value of tolerance should be 1. The **Tolerance** button should open a dialog to enter the desired tolerance values. Using different tolerance values will enable the laser tool to go through painted areas of different colors. You may check the attached "LaserTolerance.gif" to see how this tool behaves with tolerance value being 100.



The following image summarizes the behavior of the laser tool:



Preliminary Submission: You will submit an early version of your solution before the final submission. This version should at least include the following:

- SettingsFrame should be complete.
- Clicking on SettingsFrame's continue button should open Tools and Paint Frames.
- Pen and PenSize tools should be functional; i.e., we should be able to draw on the PaintFrame using pens of different sizes.

You will have time to complete your solution after you submit your preliminary solution. You can consult the TAs and tutors during the lab. Do not forget to make your final submission at the end. Even if you finish the assignment in the preliminary submission, you should submit for the final submission on Moodle. **Not completing the preliminary submission on time results in a 50% reduction of this assignment's final grade.**