TCSS 305 Programming Practicum  
Assignment 6 – Paint

# Part A Due: Friday, 22 November

# Part B Due: Friday, 29 November

Part C requirements will be published later

**Part C Due: Friday, 6 December**

### Program Description:

This assignment is designed to test your increasing understanding of writing graphical applications in Java. The key concepts are 2D graphics, mouse events, and some new Swing components. The Oracle [Swing tutorial](http://docs.oracle.com/javase/tutorial/uiswing/) provides explanations and examples which may be helpful while working on this assignment.

### GUI Appearance:

The paint program main window must have the following properties:

* The window title must be "**TCSS 305 Paint**" and the title bar must contain an icon other than the default Java icon. Be creative; choose an icon that seems appropriate to you. Use the same icon on the “About” message described later. (The screenshots show a University of Washington icon.)
* The initial window width should be 1/3 of the screen width and the initial window height should be 1/3 of the screen height and the window should be resizable.
* The GUI should use the ‘Metal’ look and feel as shown in the second JFrame code example this quarter. This has already been done for you in the provided PaintMain class.
* When the program opens the window should be centered on the screen.
* When the window is closed, the program should terminate.
* Below are some screen shots of the window in various states:

|  |  |
| --- | --- |
| *GUI with Options menu open:* | *Use matching icons on the JFrame and ‘About’ message:* |
| *Move the toolbar:* | *Tool selection in the menu and toolbar should match:* |

The window should have a menu bar with the following items.

Options

Thickness (a **submenu** containing a JSlider with the following properties:

The slider's minimum value is 0; the slider's maximum value is 25.

(If the slider is set to 0 it should behave exactly the same as if the slider were set to 1.

In other words, a thickness setting of 0 or 1 will drawn shapes 1 pixel wide.)

The slider has major tick marks (with labels) at increments of 5

The slider has minor tick marks at increments of 1

The initially selected thickness is 5. Changing the thickness affects future drawn items but not those that are already drawn.

~~---~~ (separator)

Color… A menu item which invokes a color chooser dialog

~~---~~ (separator)

Clear (removes all drawn shapes)

NOTE: this menu item should be disabled when there are no shapes to 'clear'

Tools

Line (a radio button item; selects the line tool)

Rectangle (a radio button item, selects the rectangle tool)

Ellipse (a radio button item, selects the ellipse tool)

Pencil (a radio button item, selects the pencil tool)

Help

About... (pops up a JOptionPane dialog with the following title:

**TCSS 305 Paint**

and with the following message:

**<Your Name>**

**Autumn 2024**

You may add other information to this message if you wish.

The Icon shown on this message should match the one on the JFrame, but you may use a different size icon if you wish.) This JOptionPane should pop up centered on top of the JFrame.

NOTE: The three dots shown in the Color… and About… menu items are called an ellipsis, which is a standard convention to indicate that selecting that item will open a pop up dialog.

### Components and Event Handling:

The window should contain the following components:

* A **drawing panel** in the center of the window, on which the user can draw with lines, and/or shapes. The drawing panel should listen for mouse activity and draw the appropriate item using the currently selected drawing tool. Set the drawing panel's background color to **white**. As the window resizes, the panel should grow to fill available space. The panel is initially empty, being entirely white when it first appears on screen.
* A **toolbar** (JToolBar) with four buttons in horizontal orientation in the south region of the window. The tool buttons are toggle buttons representing the tools that can be used to draw. Exactly one tool is selected at all times. The initially selected tool when the program loads should be the **Line** tool. Notice that the toolbar and the “Tools” menu contain the same commands; they should behave identically when invoked from either place. For example, selecting a tool button (such as the “Line” button) should enable the tool and cause the corresponding menu option to be selected, and vice-versa. For full credit, the behavior of each tool must be defined in your code only once, and you must be able to add tools to both the toolbar and the “Tools” menu without writing redundant code (in particular, without redundantly specifying the display names, icons, or event handlers for the tools). The tool buttons on the toolbar and in the menu must have corresponding icons – image files for icons are supplied in the project.
  + **Line tool**:   
    Draws a line on the drawing panel in the currently selected color. After selecting this tool, pressing any mouse button in the drawing panel begins line drawing by **anchoring the line's first point**. As the mouse is dragged, the line tool maintains a "hovering" line following the mouse, from the first anchored point to the mouse's current location. When the mouse button is released, the line is placed onto the panel and the "hover" stops following the mouse. This is the initially selected tool.
  + **Rectangle tool**:   
    Draws a rectangle on the drawing panel in the currently selected color. After selecting this tool, pressing any mouse button in the drawing panel begins rectangle drawing by **anchoring one of the rectangle's corner points**. As the mouse is dragged, the rectangle tool maintains a "hovering" rectangle following the mouse, from the first anchored point to the mouse's current location. When the mouse button is released, the rectangle is placed onto the panel and the "hover" stops following the mouse.
  + **Ellipse tool**:   
    Draws an ellipse on the drawing panel in the currently selected color. After selecting this tool, pressing any mouse button in the drawing panel begins ellipse drawing by **anchoring one of the ellipse's bounding box's corner points**. As the mouse is dragged, the ellipse tool maintains a "hovering" ellipse following the mouse, with its bounding box from the first anchored point to the mouse's current location. When the mouse button is released, the ellipse is placed onto the panel and the "hover" stops following the mouse.
  + **Pencil tool:**   
    Draws a freeform curve on the drawing panel in the currently selected color. After selecting this tool, pressing any mouse button in the drawing panel begins pencil drawing. As the mouse is dragged, the pencil tool draws a solid path through where the mouse is moving. When the mouse button is released, pencil drawing stops.

Implement the pencil by drawing a line between each pair of points the mouse touches. The Java2D class java.awt.geom.Path2D.Double works well for this.

**About the "Color…" menu item:**Clicking this item causes a color chooser to pop up. The color chooser should correctly indicate the current color in use. If a new color is chosen it becomes the color to use.

Note: Changing the selected color should *not* retroactively change the color of past drawn items.

Note: The initial color is “digital Spirit Purple”.

See the [UW Brand web site](http://www.washington.edu/brand/graphic-elements/primary-color-palette/) for the correct color specification.)

### Implementation Guidelines and Hints:

Implement your main drawing area as a class that extends JPanel. Attach mouse listeners to this panel to detect mouse events. Remember to handle mouse button events and mouse movement events you will need to assign 2 listeners (or add the same adapter twice). Your drawing panel class should use a cross hair cursor. The setCursor() method can be used to accomplish this.

Put your tool buttons and tool menu items into 2 button groups, so that only one of each may be selected at a time. Do *not* put the tool buttons and tool menu items into the same button group; you won’t like the result.

To synchronize the tool buttons and the tool menu items, use the Action interface and its SELECTED\_KEY

attribute. Use a separate Action for each tool button / menu item pair. Do not use a single Action with an if/else structure (or other conditional structure) to handle all tool button / menu item pairs.

You should probably store a collection of all shapes that have ever been drawn, along with each one's paint color and stroke width (you may want to write a custom class to store shapes/colors/stroke widths together).

Note that rectangles and ellipses may be drawn by dragging the mouse upward or leftward as well as downward or rightward; this will not work properly for you unless you handle it explicitly. Also note that java.awt.geom.Ellipse2D and java.awt.geom.Rectangle2D both extend the class java.awt.geom.RectangularShape; you should be able to use the same code, or very similar code, to draw rectangles and ellipses.

Your project implementation should not require an if/else or switch statement to determine the type of tool in use. It is better object-oriented design to implement a class hierarchy for handling the paint tools. This will eliminate the need for an if/else or switch statement to determine behavior for the current tool and make it easier to add another tool in future versions of the program. The hierarchy would probably include an interface, an abstract class, and some concrete sub-classes (as we saw in Assignment 4 and in the Employee example). A typical design is to instantiate one object for each tool and use the tool buttons/menu items to determine which tool should be used for drawing at any given time by setting a myCurrentTool field in the class with your JPanel.

You can (optionally) smooth the graphics on your panel by calling the following command on your Graphics2D object, which "anti-aliases" the panel (blurs the pixels):

g2d.setRenderingHint(RenderingHints.KEY\_ANTIALIASING,

RenderingHints.VALUE\_ANTIALIAS\_ON);

Keep in mind that the assignment 3 GUI included a custom icon on the JFrame; therefore, the assignment 2 GUI code provides an example for how to replace the default Java coffee cup icon on the JFrame.

Keep in mind that the assignment 4 GUI included a slider control; therefore, the assignment 4 GUI code provides an example for JSlider usage.

Place your code for the project into 2 or more meaningfully named packages. No code should be in the 'default' package. This will be part of the internal correctness grade for part B.

Submission and Grading:

It is a good idea to test your code on lab computers in SCI 106/108 or DOU 110 before your final submission.

For Part A, the appearance of **all** GUI elements must be exactly as described except as noted in this paragraph. All defaults should be shown as selected when the GUI starts (Draw color = digital Spirit Purple, Thickness = 5, Tool = Line). All menu items and submenus must be shown correctly. The program must provide the ability to draw using the **Line** tool, in the color digital Spirit Purple (that is, the ability to drag the mouse, see the line in progress, and have the line remain when the user lets go of the button). The ability to draw multiple lines is *not* required for Part A, but you are *strongly* encouraged to design it so that you can; it will save you significant time and energy on Part B. It is not necessary to coordinate the selection of toolbar and menu bar buttons. *Part A will be graded* ***only*** *on external correctness and the executive summary* - *if you have all GUI components in the right places and a user can draw a* digital Spirit Purple *line with stroke width = 5, and your program does not produce exceptions or other console output, then you are guaranteed to get full credit on Part A*.

For Part B, all features must be finished, and you will be graded on both external and internal correctness. It is possible to lose points for the same issues on both Part A and Part B; if you have external correctness problems with Part A, and they remain in your Part B submission, you will lose points for them twice.

Create your Eclipse project by downloading the assignment6-project.zip file from Canvas, importing it into your workspace, and using “Refactor” to change “username” to your UWNetID. To submit Part A, commit your project to Canvas and submit an executive summary as was done for previous assignments. Then, continue working *on the same Eclipse project* for Part B. When you are finished with Part B, commit your project to Canvas and submit another executive summary. The required filename for the Part A executive summary is “**username-assignment6-a.txt**”, and the required filename for the Part B executive summary is “**username-assignment6-b.txt**”, where username is your UWNetID. As on previous assignments, executive summaries will *only* be accepted in plain text format.

External correctness of Part A is based on the GUI’s appearance and on the ability to draw a single Spirit Purple line with the Line tool. Part A code *will not be examined* except under extraordinary circumstances (odd GUI appearance or line drawing behavior); Part A is graded *entirely* on external correctness.

The external correctness of your Part B submission will be graded on the GUI's behavior, which is observed by running the GUI, clicking various buttons, attempting to draw various shapes, and examining the result. Your GUI should match the expected layout and should be positioned, sized, and resize identically to expectations. Exceptions should not occur under normal usage. Your program should not produce any console output.

Internal correctness of Part B will be based on following the program specification, inclusion of reasonable comments, the use of meaningful identifier names, encapsulation, and the avoidance of redundancy. In addition, the output of the plugin tools will be used for Part B.

The Part A percentage breakdown is 10% executive summary and 90% external correctness.

(30% of the grade is for the ability to draw on the screen using the **Line** tool).

For Part B, the percentage breakdown is 10% executive summary, 60% external correctness, and 30% internal correctness.