a3 # assignment

Assignment 3 – Sketch Pad

School of Engineering and Technology, University of Washington Tacoma
TCSS 305 Programming Practicum, Winter 2026
Value: 5% of the course grade

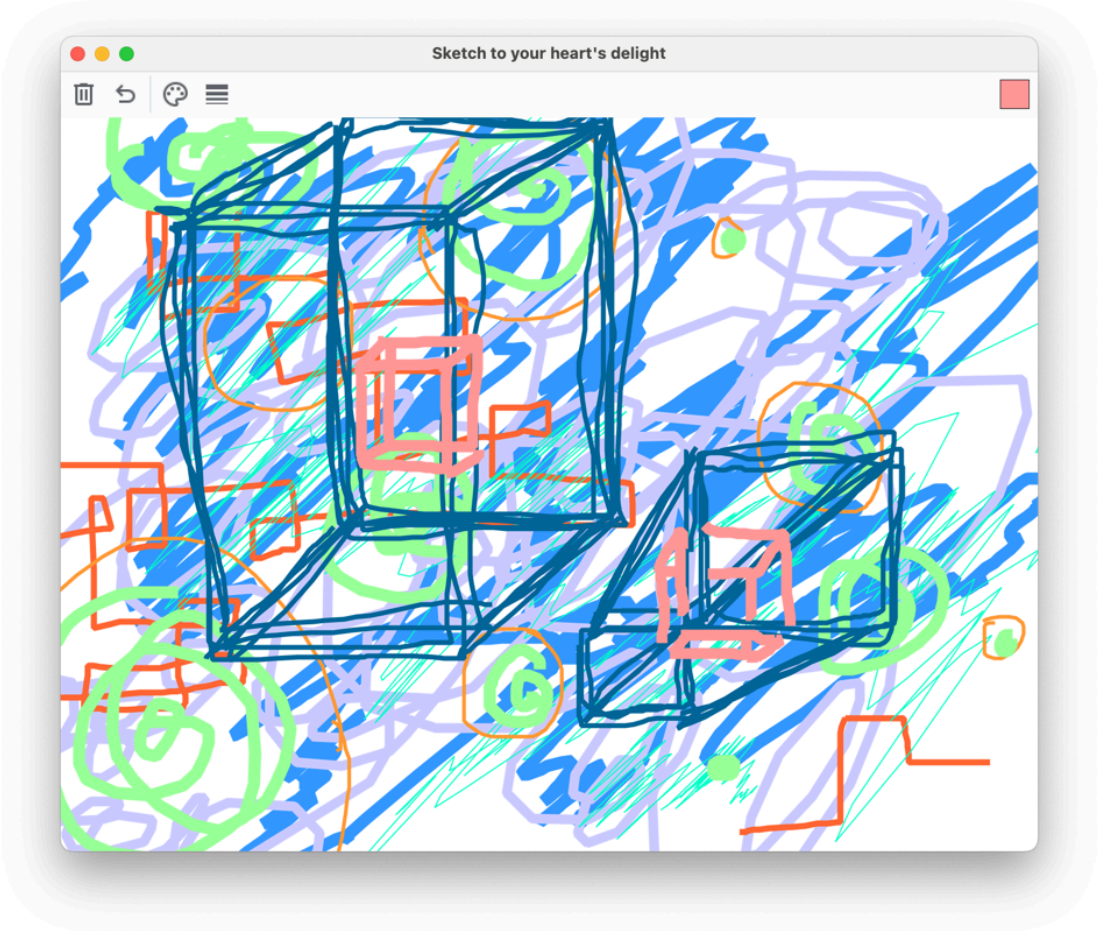
Due Date

Sunday, 15 February 2026, 23:59:59

📄 Description

In this project, you will work on a graphical user interface (GUI) application that allows you to sketch lines on a canvas. You will connect GUI events — button clicks, mouse presses, mouse drags, and mouse releases — to back-end functionality that is already implemented by your dev team.

Your task is to become familiar with the back-end API and wire up the event listeners that make the application work. You will also be able to undo a line, clear all lines, and set the lines' colors and widths.



🎯 Learning Objectives

By completing this assignment, you will:

- Understand event-driven programming in Java Swing
- Register and implement event listeners for buttons and mouse events
- Use lambda expressions and method references for concise event handling
- Work with `JColorChooser` and `JOptionPane` for user input dialogs
- Connect a GUI front-end to a provided back-end API
- Apply the Observer pattern through Java's listener architecture

---

✅ Before You Begin

Ensure you have completed:

- [x] JDK 25 installed and configured
- [x] IntelliJ IDEA installed with Checkstyle plugin
- [x] Completed Assignments 1a, 1b, 1c, and 2
- [x] Familiarity with Java interfaces and polymorphism (from Assignment 2)

📦 Project Setup

This project uses **GitHub Classroom** to distribute starter code and collect submissions.

Accept the Assignment

> **GitHub Classroom Assignment**
>
> Accept Assignment 3

1. Click the GitHub Classroom assignment link provided by your instructor.
2. **First time only:** Select your name from the course roster to link your UW identity to your GitHub username.
3. Click **Accept this assignment**. GitHub creates a personal repository for you.
4. Copy the repository URL from the green **Code** button.

Clone the Repository in IntelliJ

1. Open IntelliJ IDEA.
2. **File** → **New** → **Project from Version Control** (or **Get from VCS** on Welcome screen)
3. Paste the repository URL and choose a location.
4. Click **Clone**, then **Trust Project** when prompted.

📁 Project Structure

The starter code includes a complete GUI, back-end logic, and stubs where you will add event listeners.

```
TCSS305-a3/
├── src/
│   └── edu/uw/tcss/app/
```

```
│       ├── Application.java                          (main class - run this)
│       ├── model/
│       │   ├── ShapeCreatorControls.java              (interface - provided)
│       │   ├── PropChangeEnabledShapeCreatorControls.java  (interface -
provided)
│       │   ├── ShapeCreator.java                      (backend logic -
provided)
│       │   ├── SketcherEvent.java                     (sealed interface -
provided)
│       │   ├── UWColor.java                           (enum - provided)
│       │   └── tools/
│       │       ├── PaintTool.java                     (interface - provided)
│       │       └── PencilTool.java                    (tool impl - provided)
│       └── view/
│           ├── SketcherGui.java                       (GUI - provided)
│           ├── SketcherCanvas.java                    (canvas - add mouse
listeners here)
│           ├── SketcherToolBar.java                   (toolbar - add button
listeners here)
│           └── icons/
│               └── ColorIcon.java                    (color swatch -
provided)
├── test/
│   └── edu/uw/tcss/app/
│       └── EmptyTest.java                             (placeholder test)
├── assets/
│   ├── defaultShapes/
│   │   └── initialShapes1.txt                         (initial shapes data)
│   ├── ic_delete_24x24.png                            (Clear All icon)
│   ├── ic_undo_24x24.png                              (Undo icon)
│   ├── ic_palette_24x24.png                           (Color icon)
│   └── ic_line_weight_24x24.png                       (Line Width icon)
├── external/
│   ├── flatlaf-3.5.4.jar                              (FlatLaf look-and-feel)
│   └── flatlaf-intellij-themes-3.5.4.jar              (FlatLaf themes)
├── executive-summary.md                               (document your
submission)
└── README.md                                          (project readme)
```

Understanding the Layout

File	Location	Purpose
`Application.java`	`src/.../app/`	Entry point — run this to start the application
`ShapeCreatorControls.java`	`src/.../model/`	Interface defining the back-end drawing API (do not modify)
`PropChangeEnabledShapeCreatorControls.`	`src/.../model/`	Extends `ShapeCreatorControls` with PropertyChangeListener support (do not

File	Location	Purpose
`java`		modify)
`ShapeCreator.java`	`src/.../model/`	Back-end implementation (do not modify)
`SketcherEvent.java`	`src/.../model/`	Sealed interface defining typed events for the PropertyChange framework (do not modify)
`UWColor.java`	`src/.../model/`	Enum of UW brand colors (do not modify)
`SketcherGui.java`	`src/.../view/`	Main GUI window — composed of toolbar and canvas (do not modify)
`SketcherCanvas.java`	`src/.../view/`	The canvas panel — **add mouse listeners here**
`SketcherToolBar.java`	`src/.../view/`	The toolbar — **add button listeners here**
`ColorIcon.java`	`src/.../view/icons/`	Color swatch icon displayed in the toolbar (do not modify)
`executive-summary.md`	project root	Document your submission

Key Classes in `SketcherGui`

The `SketcherGui` class is composed of two parts:

1. **Toolbar** (`myToolBar` field) — holds buttons at the top of the window
2. **Paint Panel** (`myPaintPanel` field) — the canvas that can be sketched on

The `SketcherGui` class also has a `myShapeCreator` field, which holds the back-end logic and uses the `ShapeCreatorControls` interface.

The `ShapeCreatorControls` API

The back-end logic is encapsulated behind the `ShapeCreatorControls` interface. All drawing, undoing, clearing, and color/width changes go through this API.

`ShapeCreatorControls` Interface

The `ShapeCreatorControls` interface defines the contract for managing shape creation on the canvas.

```
void startDrawing(int theX, int theY)
```

Sets where the shape begins (the shape's initial starting point).

```
void continueDrawing(int theX, int theY)
```

Sets where the shape's next point is when creating a shape.

```
void endDrawing(int theX, int theY)
```

Finalizes the shape.

```
void setColor(Color theColor)
```

Sets the color of the shape's line.

```
Color getColor()
```

Gets the color of the shape's line.

```
void setWidth(int theWidth)
```

Sets the width of the shape's line.

```
int getWidth()
```

Gets the width of the shape's line.

```
void clear()
```

Removes all previously drawn shapes.

```
void undo()
```

Removes the most recently drawn shape.

The interface also defines a `ColorfulShape` record that pairs a `Shape` with a `Color` and line `width`.

Line = Path

In this document, when the word "line" is used, we are referring to a *path* — a freehand drawing created by dragging the mouse — not necessarily a straight line.

Extensible Design

This interface is generic enough to support the creation of other types of two-dimensional shapes. If we added rectangles, ovals, or strictly straight lines to our application, we wouldn't need to change this API other than perhaps adding a `setShape()` method. This is the power of programming to an interface.

✏️ Requirements

Four buttons and the sketch pad canvas need functionality added to them. Your code connects the GUI events to the back-end API described above.

Background Reading

- Event-Driven Programming — What event-driven programming is and how event loops work
- Swing API Basics — Java GUI history, JFrame/JPanel hierarchy, and core Swing patterns
- Swing Layout Managers — How Swing positions components in containers

The Display Updates Automatically

The `SketcherCanvas` and `SketcherToolBar` are already wired to listen for changes from the model. When your listeners call methods on `ShapeCreatorControls` (like `clearShapes()` or `setColor()`), the canvas repaints and the toolbar's color swatch updates automatically. You don't need to call `repaint()` or manually update any UI elements — just update the model and the display follows.

It is recommended that you implement these requirements in the order listed below. There are some initial shapes on the canvas upon starting the application — these help you test your Clear All and Undo buttons before needing to get the mouse events working.

You **must** use newer Java syntax such as a **lambda expression** or a **method reference** when adding action listeners to buttons. This will implicitly create an `ActionListener` object. Do not use anonymous inner classes.

> **Gen AI & Learning: Understanding Event-Driven Architecture**
>
> AI tools can generate event listener boilerplate quickly, and connecting GUI events to a back-end API is a pattern that AI handles well. However, understanding *how* event-driven programming works — the observer pattern, listener interfaces, and the flow from user interaction to method invocation — is essential for evaluating and debugging AI-generated GUI code. When AI suggests a `MouseAdapter` or lambda expression, you need to understand *why* that approach works to catch subtle bugs in event handling.

Requirement 1: Clear All Button

The Clear All button currently does nothing when clicked. Implement functionality so that clicking this button causes the sketch pad to **remove every previously drawn line**.

What to do:

- Add an `ActionListener` to the Clear All button
- The listener should call the appropriate method on the `ShapeCreatorControls` to remove all shapes

> **Guide**
>
> - Adding Event Handlers — From ActionListener classes to lambdas: the evolution of event handling
> - Introduction to Lambda Expressions — Lambda syntax, functional interfaces, and method references

> **Where to Add Listeners** ›

Requirement 2: Undo Button

The Undo button currently does nothing when clicked. Implement functionality so that clicking this button causes the sketch pad to **remove the most recently drawn line**. If there

are no lines on the sketch pad, this button should do nothing (the back-end handles this gracefully).

What to do:

- Add an `ActionListener` to the Undo button
- The listener should call the appropriate method on the `ShapeCreatorControls` to undo the last shape

---

Requirement 3: Sketch Pad Canvas

The sketch pad canvas currently does nothing when you click on it. Implement mouse event handling so that drawing works:

1. When the mouse is first **pressed** on the canvas, a new line should be started.
2. When the mouse is **dragged** across the canvas, the line drawing should be added to (continued).
3. When the mouse is **released**, that line should be finalized and no longer modified.

What to do:

- Add a mouse listener to the canvas that handles press, drag, and release events
- Each event should call the corresponding method on the `ShapeCreatorControls`

> **Guide**
>
> Handling Mouse Events — MouseListener, MouseMotionListener, MouseAdapter, and the dual-registration gotcha

---

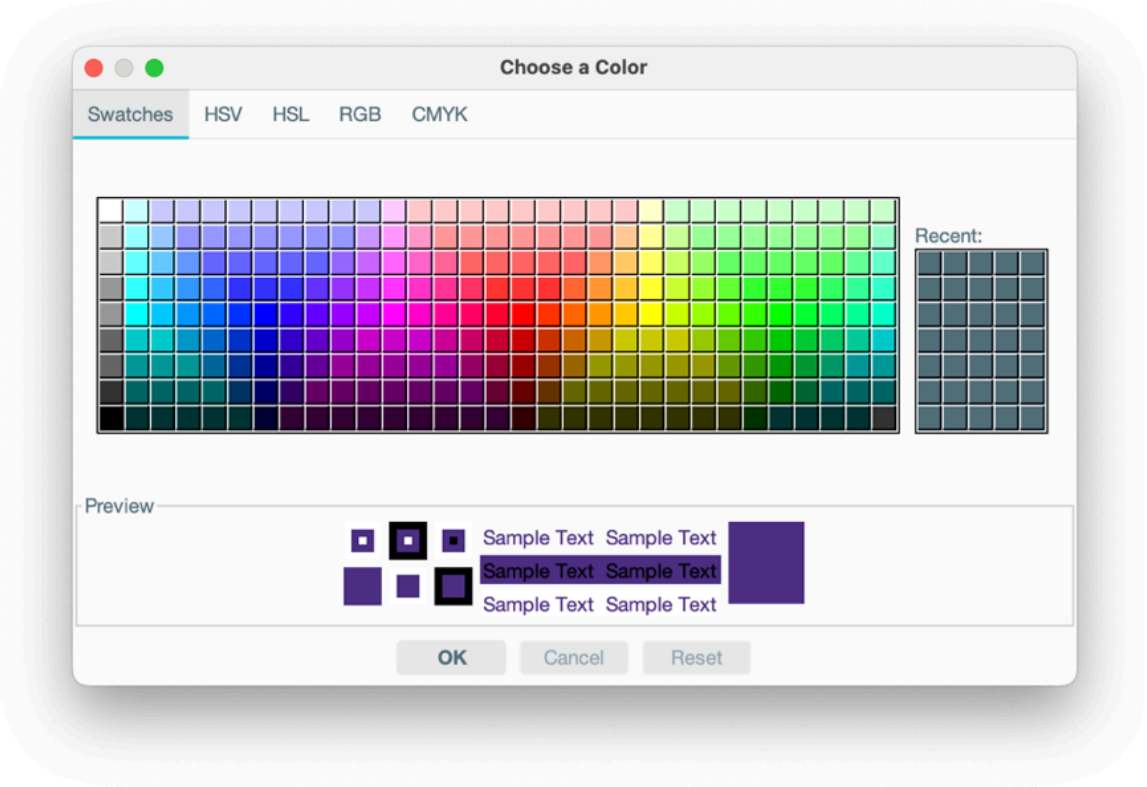
Requirement 4: Change Color Button

The Change Color button should make a color selection dialog appear. That dialog should allow a new color to be selected, and if so, the sketching color should be updated accordingly. If the color selection dialog is cancelled and no color is selected, the sketching color should remain unchanged.

What to do:

- Add an `ActionListener` to the Change Color button
- Use a `JColorChooser` to display the color selection dialog

- If a color is selected, update the `ShapeCreatorControls` with the new color
- If the dialog is cancelled (returns `null`), do not change the color



JColorChooser Documentation

- JColorChooser API
- JColorChooser Tutorial

Note

Changing the color should not affect previously drawn lines. It only changes the color for lines drawn *after* the change.

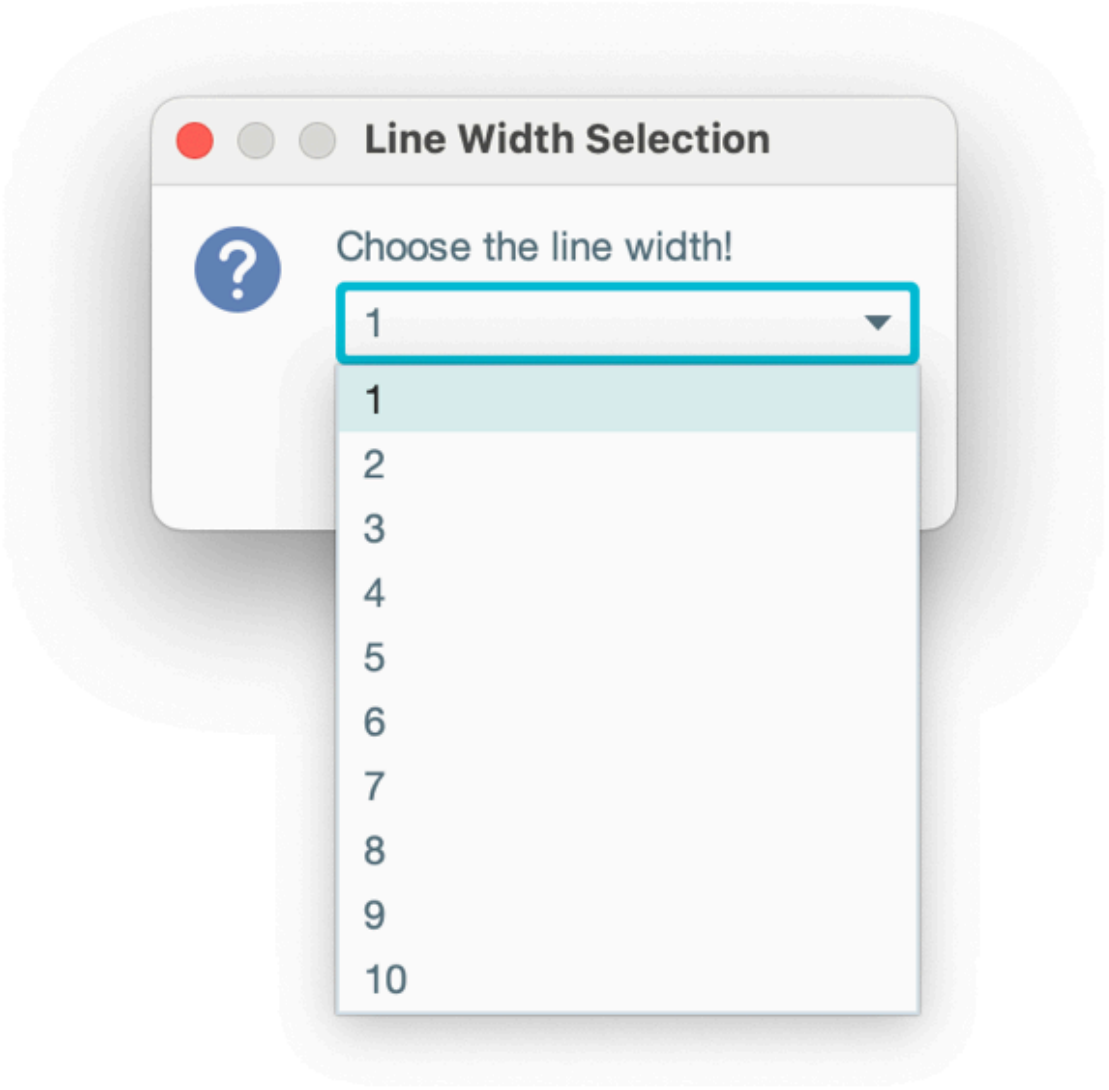
Requirement 5: Change Line Width Button

The Change Line Width button should cause a line width selection dialog to appear. If a user cancels, the sketching line width should not change. If a new line width is selected, the sketching line width should change. The line width options should be

integers representing the number of pixels. There should be ten width options: 1 pixel through 10 pixels — `SketcherToolBar` defines `MIN_WIDTH` and `MAX_WIDTH` constants for these values.

What to do:

- Add an `ActionListener` to the Change Line Width button
- Use a `javax.swing.JOptionPane` to display the width selection dialog
- Present options 1 through 10 for the user to choose from
- If a width is selected, update the `ShapeCreatorControls` with the new width
- If the dialog is cancelled, do not change the width



JOptionPane Documentation

- JOptionPane API
- How to Make Dialogs

Note

Changing the width should not affect previously drawn lines. It only changes the width for lines drawn *after* the change.

Requirement 6: Remove Initial Shapes

The application starts with some initial shapes on the canvas. These exist to help you test your Clear All and Undo buttons before you implement mouse event handling.

Before submitting, you must remove these initial shapes. In `SketcherGui.java` (line 89), change:

```
final ShapeCreator shapeCreator = new ShapeCreator(createInitialShapes());
```

to:

```
final ShapeCreator shapeCreator = new ShapeCreator();
```

Requirement 7: Executive Summary

Your project includes a file called `executive-summary.md`. This file documents your project and submission details.

The `.md` file extension stands for Markdown. Markdown is a markup language used to format plain text. You may already know a markup language — the M in HTML and XML stands for markup. (Note: markup languages are NOT programming languages but instead are tools for formatting text.)

External Resource

Markdown Cheat Sheet

Edit the `executive-summary.md` file to personalize it for your submission. Carefully read all the text found inside of the enclosing square braces `[]`. Remove all this text (and square braces), replacing it with your own specific details about the project.

📚 Guide Reference

The following guides support this assignment. See the A3 Guides page for when to read each one.

Guide	Description
Event-Driven Programming	What event-driven programming is, the event loop, and the OS role
Swing API Basics	Java GUI history (AWT → Swing → FX), JFrame/JPanel hierarchy
Swing Layout Managers	FlowLayout, BorderLayout, GridLayout, and composite layouts
Adding Event Handlers	ActionListener evolution: classes → inner classes → lambdas → method refs
Handling Mouse Events	MouseListener, MouseMotionListener, MouseAdapter, and coordinates
Introduction to Lambda Expressions	Lambda syntax, functional interfaces, and method references
Checkstyle Rules Reference	Look up specific Checkstyle violations and how to fix them
IntelliJ Inspections Reference	Look up specific IntelliJ warnings and how to fix them
Linters and Code Quality	Why linting matters and how to use Checkstyle and IntelliJ Inspections

---

🚀 Submission and Grading

Important

Start early — event-driven programming requires careful understanding of how listeners connect to back-end logic.

Please see the rubric in Canvas for a breakdown of the grade for this assignment.

Git Workflow Required

For this assignment, you must work on a **development branch** and submit via **pull request**.

Guide

Git Branching and Pull Requests — How to work with branches and submit via pull request.

Submitting Your Work

1. **Ensure all your work is committed** to your `dev` branch
2. **Push your dev branch** to GitHub:

   In IntelliJ: Click **Git** → **Push**, then click **Push**

   Or from the terminal:

   ```
   git push -u origin dev
   ```

3. **Create a Pull Request** on GitHub:
   - Go to your repository on GitHub.com
   - Click **Pull requests** → **New pull request**
   - Set base: `main` and compare: `dev`
   - Add a descriptive title and summary of your changes
   - Click **Create pull request**
4. **Merge the pull request** to main
5. **Verify:** Visit your repository on GitHub.com and confirm your changes appear on the `main` branch

Note: You can commit and push multiple times. Only your final state of `main` at the deadline will be graded.