CS193J
Winter, 2001-02

Handout #16

HW 2: JavaDraw

Due Midnight ending Tue Feb 19th

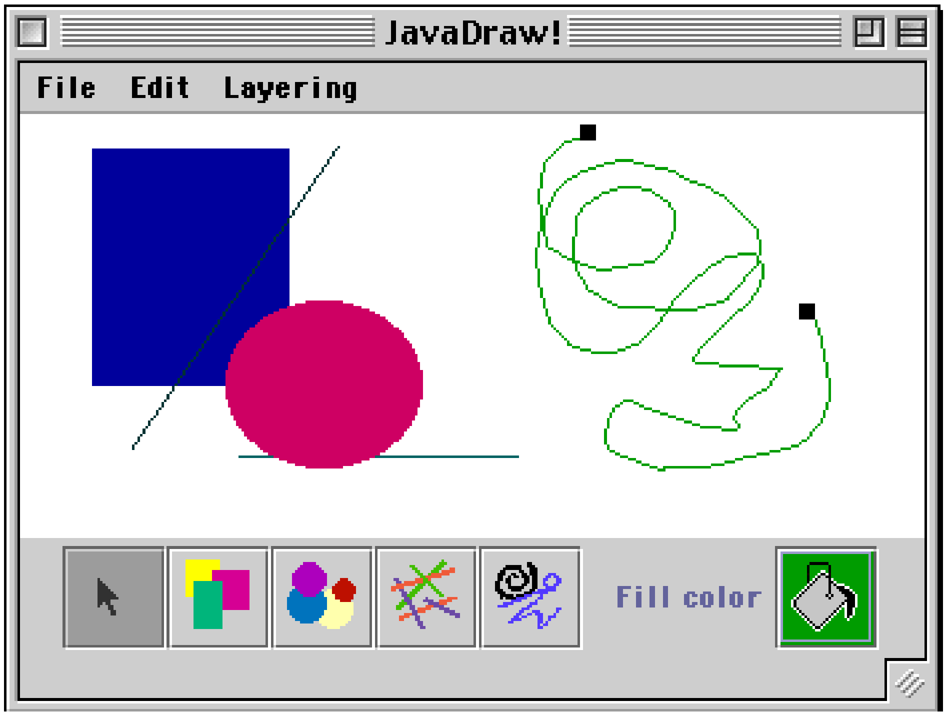
(Thanks to Julie Zelenski for creating this assignment.) Your next assignment will be to construct a simple drawing program that allows the user to draw shapes in various colors, cut and paste shapes, and save and load drawings to disk. In this handout, we will stick to discussing the functional requirements of the assignment. Design hints and development suggestions are given in a separate handout. Be sure to thoroughly read both handouts so you understand both the functional specification and our design expectations. As always, if you find something unclear, please ask! We will have the usual extra office hours in the days before the due date – see the course page.

This assignment will combine two of major OOP themes: using OOP GUI components and the OOP inheritance problem of factoring several related classes to make a little hierarchy. For the Swing part of the assignment, we have seen many of the relevant classes in lecture, but for some classes, you will need to hunt through the Sun docs. The starting project gives you a good headstart on the mundane aspects of setting up the user interface so you don't have to spend too much time on these tedious aspects.

Overall, this assignment is more involved than your last one. Developing a good object-oriented design is an incremental process that often takes several iterations until you settle on a quality solution. Don't handicap yourself by trying to throw the program together at the last minute!

JavaDraw basics

JavaDraw is a simple draw program. In its final form, it has just one window that contains a drawing canvas and a tool bar that supports the creation of rectangles, ovals, lines, and scribbles in various colors. A selection tool allows the user to select, move and resize shapes. There are menu commands to cut, copy, and paste shapes, rearrange layering, and save and load shapes from files.



What you start with

The above description and screenshot may seem daunting—however, don't panic, you won't have to start from scratch. With graphical programs, even more so than with others, it is particularly effective to learn by taking an existing program and extending it as a means of gentle introduction to a complex set of classes. If you compile and run our starting program, you'll find that it can already draw rectangles, select and deselect a rectangle, and move or resize the selected rectangle.

Your job will be to first read our code to observe the basic usage of the GUI classes and then move on to experimenting with additions, gaining confidence and deeper understanding as you go. You may begin with a bit of a "monkey see, monkey do" approach but your skill grows into a more sophisticated mastery over the course of the assignment. You avoid that frustrating initial struggle to just get anything working and you have a good design in place to guide you. Since you start with a working program, if you make small modifications as you go, you can continue to keep a working program each step along the way, which will simplify the testing and debugging work.

In our starting project, there is a JavaDraw main class that constructs the user interface. It creates the canvas window and its internals and installs the menus and their handlers. There is a Toolbar class that configures the radio group of buttons with images for the tools and the current color button that brings up the color chooser dialog. We also give you a starting implementation of the DrawingCanvas and Rectangle classes that handles simple rectangles.

What you need to do

The users want more and your mission is to start trying to satisfy their demanding requests! There are five features that you are going to add to the program:

1. *New shapes.* The most major extension that you will be making is to support three new shapes. The program as given only supports rectangles, you need to extend it to draw ovals, straight lines, and "scribbles" (a squiggle of connected lines that trace a path).
2. *Layering.* As shapes are created, they are added on top any previous shapes, overlapping and covering up those shapes below. In our starting implementation, there is no way to re-arrange the ordering after a shape has been created. The Layering menu contains unimplemented "Bring to front" and "Send to back" commands. These commands need to be implemented to move the selected shape on top or beneath all other shapes.
3. *Colors.* The existing implementation assumes all shapes are filled in dark gray. You will extend the program to create each new shape in the color currently selected on the toolbar. You will also update the toolbar to reflect the current color whenever a shape is selected and allow the user to change the color of the selected shape.
4. *Edit operations.* The Edit menu has unimplemented cut, copy, paste, and delete commands. You will implement operations to store a copy of a shape on the clipboard, paste the clipboard contents, and delete shapes. The clipboard contents will be handled via object cloning.
5. *File operations.* The File menu contains clear, load, and save commands that also require your implementation. The clear command just removes all shapes from the canvas. The save command allows you to store a drawing to a disk file and the load command loads a previous saved drawing. The file operations will be done using object serialization.

(We’ve covered the material for 1-3 already, so you can start with those. We’ll be done with the material for 5 and 6 in one more lecture, so you can wait on those.)

A little more detail

In general, the program is expected to behave in the common and obvious ways that basically all drawing applications do (MacDraw, PowerPoint, Diagram, and so on). Probably all of you are familiar with at least one of these applications and thus you will have an idea of what the goal is. You can also examine the behavior of our starting program to see how create, select, move, and resize behave. The requirement summary at the end of this handout goes into painstaking detail about the required behaviors that can help if you aren't sure of the nuances.

Here is a quick summary of some of the basic behaviors to get you started: The tool buttons allow the user to select the current tool mode in effect. The tool mode determines what happens when the user clicks and drags across the canvas. If the toolbar has a shape tool selected, clicking and dragging on the canvas draws out a new instance of that shape. If the toolbar has the arrow tool selected, mouse actions on the canvas will select, move, and resize shapes. Clicking on a shape selects it and a selected shape is drawn with resize knobs. Clicking and dragging on a shape moves it. Clicking and dragging on a resize knob resizes the shape. All shapes are filled, the color is set at creation and can later be changed with the toolbar's color button. The layering commands move shapes above or below others. Cut, copy, and paste allow you to store and retrieve duplicate copies of a shape on the clipboard. The load and save commands allow you to save a drawing to disk and reload it later. The assignment does not require support for advanced features like multiple shape selection, undo, printing, framed shapes, multiple document windows, scrolling, and so on.

Input and output

Again, you will need to use a bit of file I/O, yet we still haven't gotten to talking about exceptions which are usually heavily in the stream classes. As we did for hw1, we will provide you with wrapper classes to help smooth this over. The `SimpleObjectReader` class can open a file for reading and has methods to read objects from it one by one and rehydrate them. It expects the objects were serialized to the file using a `SimpleObjectWriter`. This class opens a file for writing and can write serializable objects to it. The provided source files are documented (in javadoc format) with more details on the methods and usage for each class.

Requirements summary

Like we did for HW2, a summary list we constructed to help you manage the details:

General

- Your source files should be easily readable on UNIX— i.e. end-of-line characters should be proper, lines shouldn't wrap in obnoxious places, etc. This is of particular importance to those of you moving your files to Solaris from elsewhere.
- The submitted project should include all necessary source files and images. We should be able to issue the command `javac *.java` and all files should compile cleanly (i.e. with no warnings).
- Your main class should be named JavaDraw and should require no command-line arguments. After compiling, we should be able to run your program with the command `java JavaDraw`.
- The program should run as a Java application, not an applet.

Selection tool

When the arrow tool is the currently selected tool mode on the toolbar:

- A click causes the topmost shape under the mouse to become the currently selected shape. Any previously selected shape is deselected. The currently selected shape is drawn with resize knobs on its corners. Clicking on a shape that is already selected makes no change.
- Clicking on the canvas but not directly on a shape deselects any previously selected shape, leaving the selection empty.
- Clicking and dragging on a visible knob of the selected shape resizes the shape, allowing it to shrink and grow. Resizing affects some shapes a little differently (see descriptions below).
- Clicking on a shape (but not within a resize knob) and dragging causes the shape to move with the mouse. The shape stays the same size but is translated around the canvas.

- There is immediate visual feedback when selecting or deselecting a shape as well as continual updates during moving and resizing.

Rectangles

- When the rectangle is the currently selected tool mode on the toolbar, clicking and dragging on the canvas creates a new rectangle that resizes with the user's drag. The user can drag out the new shape in any direction (i.e. not just down and to the right). The new rectangle appears on top of any previous shapes and is filled with the toolbar's currently selected color.
- Moving a rectangle translates the coordinates of its bounding box in the obvious way.
- Resizing a rectangle moves the selected knob to a new location while the opposite diagonal knob remains "anchored" in place. A rectangle can be resized smaller or larger in either or both dimensions.

Ovals

- When the oval tool is the currently selected button on the toolbar, clicking and dragging on the canvas creates a new oval that resizes with the user's drag. The new oval appears on top of any previous shapes and is filled with the toolbar's currently selected color.
- An oval can be thought of as transcribed within a rectangular bounding box. The behavior of that bounding box is quite similar to the standard rectangle shape.
- An oval has four resizing knobs, one on each corner of the bounding box.
- For a click to select an oval, it must be within the oval's filled region. A click within the bounding box but outside the filled area (i.e. in one of the corners) does not select the oval.
- Moving an oval translates the coordinates in the obvious way.
- Resizing an oval moves the selected knob to a new location while the opposite diagonal knob remains "anchored" in place. This changes the oval's bounding box, which changes the dimensions of the oval. An oval can be resized smaller or larger in either or both dimensions.

Lines

- When the line tool is the currently selected button on the toolbar, clicking and dragging on the canvas creates a new straight line that resizes with the user's drag. The new line appears on top of any previous shapes and is drawn in the toolbar' s currently selected color.
- A line can be thought of as a diagonal line across a rectangular bounding box. The behavior of that bounding box is quite similar to the standard rectangle shape.
- A line has two resizing knobs, one on each end of the line segment.
- For a click to select a line, it must be fairly close to the line segment. A click within the bounding box that is not near the line (i.e. in an opposite corner) does not select the line.
- Moving a line translates its coordinates in the obvious manner.
- Resizing a line moves the selected knob to a new location while the other knob remains "anchored" in place. This changes the line's bounding box, which changes the length and angle of the line. A line can be resized smaller or larger in either or both dimensions.

Scribbles

- When the scribble tool is the currently selected button on the toolbar, clicking and dragging on the canvas traces out a path on the canvas following the mouse movement. The new scribble appears on top of any previous shapes and is drawn in the toolbar's currently selected color.
- A scribble can be thought of as a sequence of points connected by line segments. The minimum and maximum coordinates along the path in both dimensions define the scribble's rectangular bounding box.

- A scribble has two resizing knobs, one on each end of the scribble (i.e. the first and last points on the path)
- For a click to select a scribble, it must be reasonably close to a segment of the path. A click within the bounding box that is not near a path segment does not select the scribble.
- Moving a scribble translates its coordinates in the obvious manner.
- Resizing a scribble works differently than other shapes. Clicking and dragging one of the resize knobs doesn't scale the entire scribble, but instead allows you to extend the path in that direction with additional points. The end of the scribble path follows the mouse drag and lengthens the path, potentially enlarging its bounding box. There is no way to shrink or shorten a scribble path.

Layering

- The "Bring to Front" command moves the selected shape to the top so that it is drawn above all other shapes and will be the first to receive mouse clicks. The ordering of all other shapes remains the same. This command has no effect when there is no selected shape.
- The "Send to Back" command moves the selected shape to the bottom so that it is drawn below all other shapes and will be the last to receive mouse clicks. The ordering of all other shapes remains the same. This command has no effect when there is no selected shape.

Colors

- Whenever a new shape is selected, the color button on the toolbar updates to show the color of the selected shape.
- Clicking on the color button brings up the color chooser dialog that allows the user to select a new color. After the user chooses a color and dismisses the dialog, the currently selected shape (if any) changes to the new color.

Edit operations

- The clipboard operations must be implemented by object cloning. Each shape must be capable of cloning itself to store onto the clipboard. We are not interfacing with the system clipboard which has its own complications. Our "clipboard" is simply a copy of the most recently cut or copied shape.
- The "Copy" command stores a copy of the selected shape onto the clipboard, replacing any previous clipboard contents. There is no change to the shape or the canvas. The copy command has no effect when there is no selected shape.
- The "Cut" command copies the selected shape onto the clipboard and deletes the shape from the canvas. After a cut, there will be no selected shape. The cut command has no effect when there is no selected shape.
- The "Paste" command adds the shape on the clipboard to the canvas. The newly pasted shape becomes the selected shape. The pasted shape should be a duplicate of the last shape that was cut or copied (e.g. same shape type, location, size, color). The same clipboard contents can be pasted multiple times. It is a nice touch (but not required) if a subsequent paste is offset slightly from the previous so the identical shapes don't pile on top of one another. The paste command has no effect if the clipboard is empty (i.e. no cut or copy has been performed).
- The "Delete" command deletes the selected shape from the canvas. It does not change the clipboard contents. After a delete, there will be no selected shape. This command has no effect when there is no selected shape.

File operations

- The "Clear All" command deletes all shapes, returning to a blank canvas. There will be no selected shape.

- The file operations must be implemented by object serialization. Each shape must be able to read and write itself to a stream. We provide simple classes that wrap the object streams to simplify the I/O and handle exceptions for you.
- The "Load" command brings up a file chooser dialog that allows the user to select a file to load. The program should open the file, read the shapes, and replace the current contents of the canvas with the shapes from the file. There should be no selected shape after a successful load. If the file cannot be read or doesn't contain any valid shapes, the current contents of the canvas are left unchanged.
- The "Save" command brings up a file chooser dialog that allows the user to select a filename for saving. All of the shapes in the canvas should be saved to the file. The contents of the canvas are not changed in any way. If the file cannot be written, no error is reported.
- The "Quit" command exits the program. It does not have any safety checks to ask the user to save changes before quitting or any such niceties; it just immediately exits. Closing the window also exits (The code for this case is already in the starter file.)

A few random details

The starting project uses Swing components and thus will require an environment with Swing GUI support – see the course tools page for help on running Swing on the various platform. Running a Swing app on our leland Solaris machines sometimes may spew messages about missing fonts. These are harmless and can be ignored. Depending on your window manager on leland, some windows may show up with the slightly wrong appearance. This is an interaction bug between the Swing implementation and the window manager, so it's nothing to do with your code.

Grading

Functionality will be tested by verifying that your draw program can perform all the required operations on all four types of shapes, redraw correctly as shapes are moved, resized, change color, etc., can cut/copy/paste, load and save files, and so on. Design will be weighted somewhat more on this assignment than the last, since that is where much of your effort will be going. Design will consider how well you succeeded at constructing a sensible hierarchy, factoring the common data and code, and designing objects that provide for good abstraction and encapsulation.

Getting started

As usual, there are starter files available off the course page. There's a lot of basic stuff done for you, so you'll need to spend a little time familiarizing yourself with the provided code.

No matter where you do your development work, when done, it is your responsibility to ensure your project will compile and run properly on the leland workstations. All assignments will be electronically submitted there and that is where we will compile and test your project. We recommend moving your code over a day or two in advance to give yourself plenty of time to test and resolve any problems instead of trying to deal with everything in a last-minute panic.