Project 1 Documentation

08/24/18

Andrew Eissen

CMSC 405 6381 (2188)

Section I: Deliverables

As per the Project 1 rubric requirements, the author's Project 1 submission includes all relevant documentation and files necessary to the running and interpretation of the program. More specifically, the package includes this documentation file in `.pdf` form, a set of three (3) Java source files (namely `Driver.java`, `Application.java`, and `ImagePanel.java`), and a `.zip` file of the relevant HD screenshot images for ease of external viewing. The author is aware that the inclusion of images in a Microsoft Word or `.pdf` file often results in grainy thumbnail images that are difficult to see, and has thus seen fit to include the relevant screenshots in the package for ease of viewing.

Section II: Design and test plan

The author began the project construction process with the development of a hand-coded user GUI incorporating a series of buttons and panels. As the project design rubric required the implementation and display of three separate method-created images but gave no specifics as to the manner in which these images were to be displayed, the author elected to pursue a different approach than that exemplified in the template files package. Rather than display the images in one panel, he designed a separate `JPanel`-extending `ImagePanel` class and built three separate instances of that class for display in the GUI as panels. Each instance was provided a parameter consisting of a separate two dimensional array containing shape and `rgb` color information from which the image was formed and displayed to the user. Furthermore, to allow the user to more easily follow the animation, a status log was provided in the bottom panel of the GUI, logging messages related to the transformations being performed as they occurred along with any error messages encountered during the program's running.

Though the GUI design remained constant across the project lifetime for the most part, the codebase implementation changed several times over the course of the assignment, with certain methods originally included in the templates package shifted around between classes and restructured

or rewritten as needed to make the transformation progression easier to follow in the code for both the author and the user. For example, each of the `ImagePanel` instances created to display the three images originally possessed their own `AffineTransform` objects, which made coordinating transformations difficult and required lots of duplicate code. Rather than allow each image to transform itself in parallel with the other images, the author moved the relevant code to the main `Application` class instance. This instance coordinated all image transformations using a single `AffineTransform` object which was concatenated with each of the separate images' own individual transform instances in their `paintComponent` methods.

As far as the author's test plan was concerned, once the program had begun displaying the proper transformations as required, the author began commenting out various parts of the code to isolate selected functionality and methods as needed. Specifically, each transformation was examined separately in detail to ensure that it was manipulating the images as expected and in accordance with the rubric requirements and user expectations. This was done via the use a `boolean` constant called `DEBUG` which controlled the display and application of a set of unit testing processes. Foremost among these was `Application.testTransformation`, a method which allowed the author to test each transformation separately as seen in the following "Test cases" section of this document.

This frame-by-frame debugging approach paid off for the author in the unit testing phase. Prior to testing, the images appeared to be rendering correctly, allowing the author to switch focus to documenting and optimizing sections of the codebase. However, upon testing each operation, the author noticed that the images were not being transformed properly despite a lack of actual errors. Testing indicated that the transformations were occurring in accordance with `Graphics2D`'s default coordinate system, in which increasing y-axis values extend downward towards the bottom of the screen (Flanagan, 1999). The images were being moved correctly but were not making sense from the

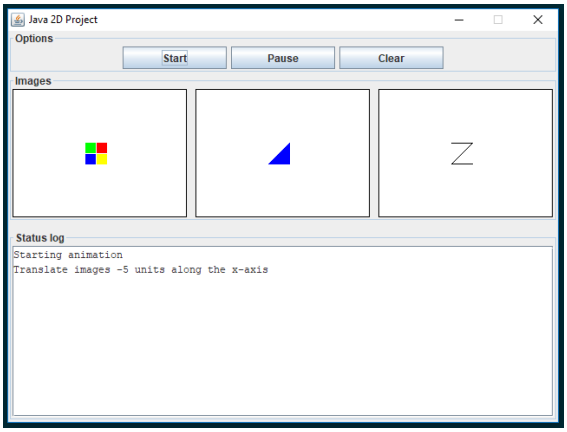
user's visual perspective, given that in most conventional coordinate systems, increasing y-axis values are arranged as they go up the screen. As a result of this testing, the author was able to implement a workaround that flipped the images horizontally to ensure operations appeared properly to the user.

Section III: Test cases

As per the Project 1 design rubric and the author's debugging test plan as described above, each of the methods (taken in context to be each of the six required transforms) was to be tested individually. Furthermore, the author has included a pair of test cases related to so-called "illegitimate" presses of the user GUI buttons, i.e. cases wherein the "Start" button is pressed while the animation is running or wherein the "Pause" button is pressed while the animation is already stopped. Again, as per the Project 1 rubric instructions, all transformation operations were undertaken on top of the images as they appeared following all previous transformations. At no point does the program revert to the original images before applying new transformations until all the required operations in the transformation cycle/progression have been completed. Only then does the program loop.

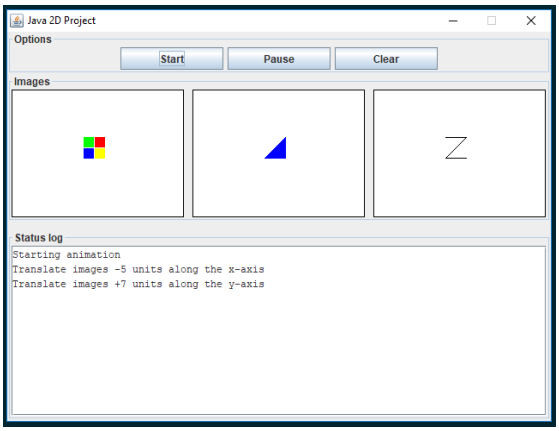
Transformation	**Method tested**	**Testing process/method**	**Result of testing**
Translation: -5 on x-axis	AffineTransform.translate()	translate(-5.0, 0);	Images moved to left
Translation: +7 on y-axis	AffineTransform.translate()	translate(0, 7.0);	Images moved up
Rotation: 45° counter	AffineTransform.rotate()	rotate(45 * Math.PI / 180.0);	Images rotated left
Rotation: 90° clockwise	AffineTransform.rotate()	rotate(-90 * Math.PI / 180.0);	Images rotated right
Scale: 2x on x-axis	AffineTransform.scale()	scale(2.0, 1.0);	Image widths expanded
Scale: 0.5x on y-axis	AffineTransform.scale()	scale(1.0, 0.5);	Image heights shrank in half
Button: *Start* after begin	Application.buttonHandler()	buttonHandler(false, "Starting animation", "Press \"Pause\" to pause", "start");	Prompt message displayed
Button: *Pause* after pause	Application.buttonHandler()	buttonHandler(true, "Pausing animation", "Press \"Start\" to resume", "stop");	Prompt message displayed

Test Case 1: Translation -5 on x-axis



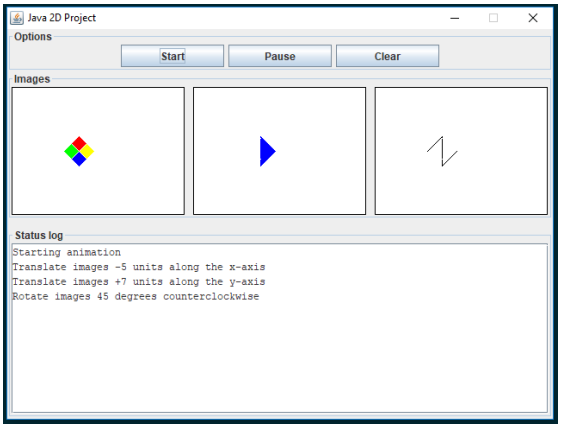
The first transformation operation undertaken in the six-operation progression was the horizontal translation of the images along their individual x-axes -5 pixels, or five units to the left in the case of the coordinate system employed. The operation occurred as expected, as demonstrated in the screenshot.

Test Case 2: Translation +7 on y-axis



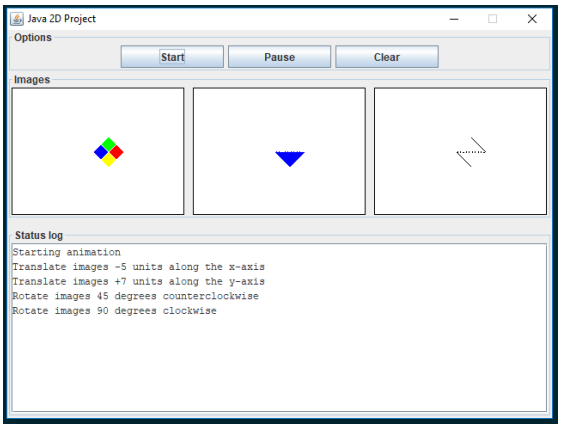
Related to the first test case, the second transformation operation undertaken was the vertical shift of the images along their y-axes +7 pixels, or seven units to the top. Again, the operation occurred as expected, with all images moved upwards in accordance with the logical 2D coordinate system expected by the user.

Test Case 3: Rotation 45 degrees counterclockwise



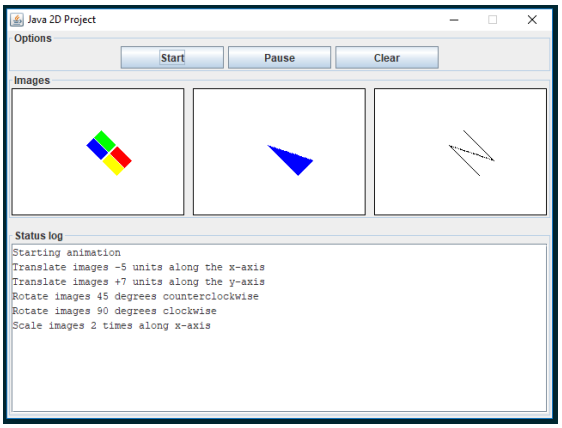
Once translated from their initial origin points in the center of the `ImagePanel` instances, the images were then rotated 45 degrees counterclockwise, calculated and formulated in the codebase as the value of `45 * Math.PI / 180.0`. As expected, the images are rotated to the left in the proper manner.

Test Case 4: Rotation 90 degrees clockwise



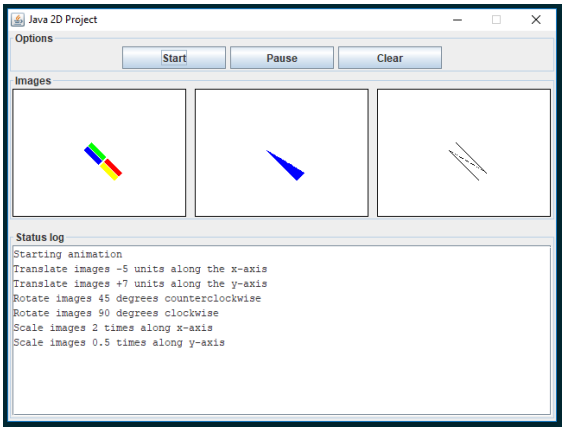
As above, once rotated to a 45 degree angle in the counterclockwise direction, the images were rotated again, this time to the right in a clockwise direction a total of 90 degrees. In the codebase, this was calculated from `-90 * Math.PI / 180.0`. The images are rotated successfully, as seen in the screenshot.

Test Case 5: Scale 2 times on x-axis



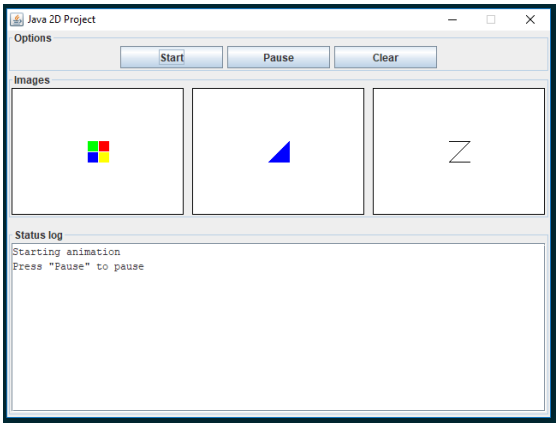
As the project rubric stated that images were not to be reset post-transformation, the two scale operations were compounded upon the images as they had been rotated in the previous cases. The first of these scale operations expanded the images' x-axis components by a factor of two, in effect expanding their widths.

Test Case 6: Scale 0.5 times on y-axis



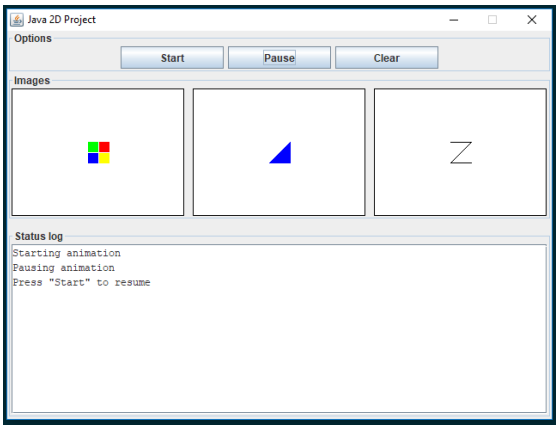
As per the aforementioned logic, the second scale operation was undertaken on the "fattened" images previously scaled in the x-axis direction. This scale operation instead worked on their height aspects, scaling them down to a factor of 0.5 on the y-axis. This effectively shrunk them in height.

Test Case 7: Pressing "Start" while running



While not required functionality, the author was bugged by the fact that pressing the "Start" button while the animation was running did not produce any message in the status log. Thus, he added a piece of functionality that displays a prompt indicating that the user should press "Pause" to stop the animation.

Test Case 8: Pressing "Pause" while paused

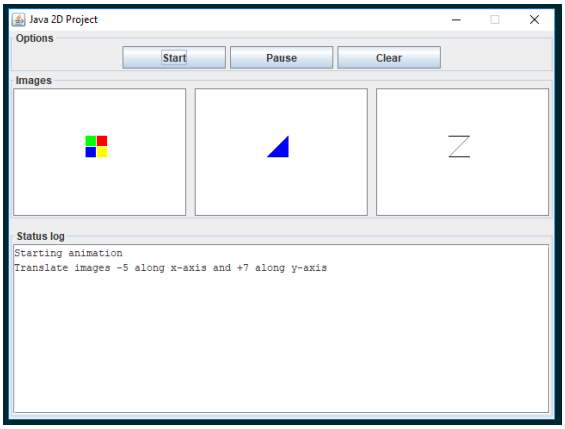


Similar to above, the author was bugged by the fact that pressing the "Pause" button while paused did not log any error messages in the status log. As such, he implemented some functionality that prompts the user to press the "Start" button to resume the animation rather than the "Pause" button.

Section IV: Default program run-through

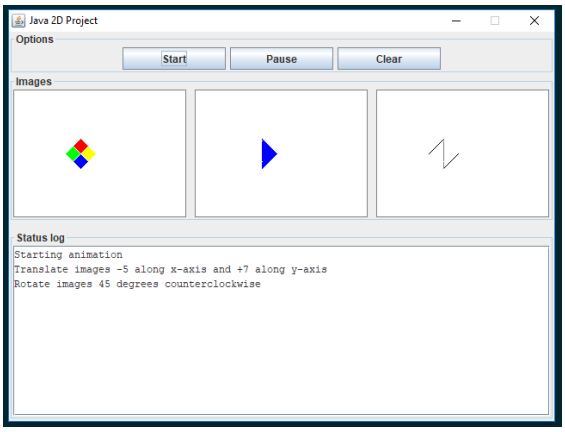
As per the Project 1 rubric requirements, the author has included a set of screenshots related to an example of a successful run-through of the program without any of the debugging functionality engaged. In the standard “production” version of the program, only four frames are displayed per the animation sequence, beginning first with the pair of translation operations, continuing on to the independent rotation operations, and ending with the pair of simultaneous scale operations. Apart from some minor status logging changes, nothing cosmetic is affected by engaging or disengaging debugging mode, as illustrated in the following screenshots.

Frame 1: Dual translation transformations



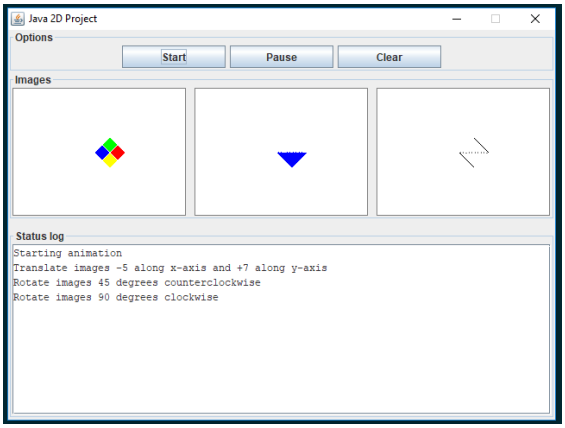
The default version of the program begins with the simultaneous translation of the images to the (-5, 7) position on the coordinate system employed by the program, in accordance with the rubric requirement that these operations occur in the same frame. As expected, the images are shifted up and to the left.

Frame 2: Rotation 45 degrees counterclockwise



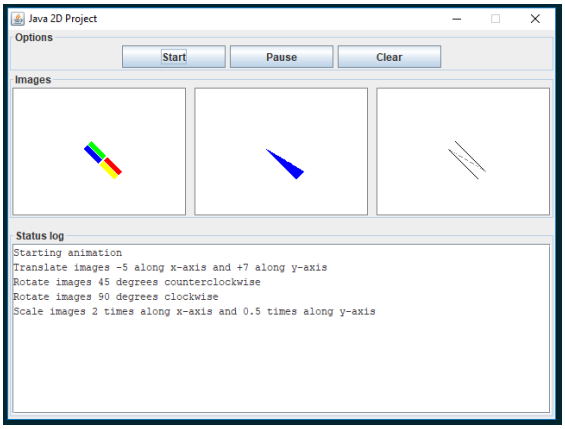
As per the debugging test cases preceding this section, the rotation operation moving the images in a counterclockwise direction 45 degrees is undertaken separately from any other operation. As expected based on the coordinate system used in the program, the images are shifted to the left 45 degrees.

Frame 3: Rotation 90 degrees clockwise



As evidenced in the preceding example and the test cases of Section III, this transformation operation is undertaken separately from the other operations required by the rubric. It rotates the images to the right 90 degrees from their previous position, moving them in accordance with the coordinate system.

Frame 4: Dual scaling transformations



As with the first frame included above, the default production version of the program without any debugging functionality activated combines the pair of scale operations together, simultaneously expanding the images' widths on their x-axes while shrinking their heights along their tilted y-axes.

Section V: Lessons learned and potential improvements

Though the author has had significant experience writing in Java up to this point, he had difficulty wrapping his head around the specifics of the `Graphics2D` paradigm for most of the project's duration. The experience has opened a new set of doors in terms of visualizing the manner in which images may be dynamically manipulated by a computer. For example, the author had never heard of an affine transform prior to this project and the week's reading. The manner in which a developer must think in order to create an animation was utterly foreign to the author, and required him to stretch his brain around the concept for some time prior to implementation. All things considered, he believes he learned much from the process concerning how to approach animation from a general perspective, knowledge that can be applied to the remaining projects of this course regardless of the languages they are to be written in or the paradigms they are to incorporate.

As far as potential improvements are concerned, the author is unsure about a number of methods implemented in the program. For example, a major concern is the manner of the `ImagePanel` `paintComponent` method's implementation. The author is unsure of whether or not the method was undertaken properly according to best coding practices, or even if it represents a case of clean, optimized code. As it currently exists, it represents the best the author can possibly produce

given his current grasp (or lack thereof) of the material. If provided more time to spend on the project, particular emphasis would have been placed on determining a better way to implement a switch of the coordinate system while preventing images from appearing inverse or mirrored to the user in the GUI.

Further, the author would have liked to spend more time messing around with the `Timer` class and its components, particularly to assist in adding a "reset" button. The author had initially intended to add such functionality to the buttons panel, but was unable to dedicate the time to this optional functionality due to unforeseen complications arising from the implementation of the required transformation operations. If this project had been one iteration of a project series spanning the length of the class, like the four-part CMSC 335 SeaPort project series, the author would have likely added such helpful functionality at a later date.

References

Flanagan, D. 1999. Graphics with AWT and Java 2D. In *Java™ Foundation Classes in a Nutshell: A Desktop Quick Reference* (chapter 4). Retrieved from https://docstore.mik.ua/orelly/java-ent/jfc/ch04_03.htm

Gallardo, R., Hommel, S., Kannan, S., Gordon, J., & Zakhour, S.B. 2014. Coordinates. In *The Java Tutorial: A Short Course on the Basics (6th Edition)*. Retrieved from https://docs.oracle.com/javase/tutorial/2d/overview/coordinate.html