Group Project: A Command-Line Vector Graphics Software

Department of Computing

September 20, 2021

Deadlines

1. Archive the source code, the tests, the presentation slides, and the final report in PDF into a ZIP file, and submit it on Blackboard before or on November 18, 2021.
2. Submit the design review report on Blackboard before or on November 23, 2021.

1 Overview

The goal of this project is to develop a Command LinE Vector graphIcs Software, or Clevis in short, in Java. Users should be able to create and manipulate vector graphics containing shapes like line segments, circles, rectangles, and squares using Clevis.

2 Requirements

The Clevis tool should provide a command line interface (CLI) to its users, and some of the requirements for the interface are as the following, where all numeric values are of type double. Note that all floating point numbers should be rounded to 2 decimal places when printed out.

[REQ1] (4 points) All the operations executed during a Clevis session should be logged into two files. The first file is in HTML format, where the commands are recorded in a table and each row of the table has two columns, one for the operation index and the other for the operation command. The second file is in plain TXT format, where the commands are recorded in execution order and each line contains one command. The names of the two files should be input as parameters when launching the Clevis tool, as in “java Clevis-html log.html -txt log.txt”.

[REQ2] (1.5 points) The tool should support drawing a rectangle.
Command: `rectangle` *n x y w h*
Effect: Creates a new rectangle that has a unique name *n*, whose top-left corner is at location (*x*, *y*), and whose width and height are *w* and *h*, respectively.

[REQ3] (1.5 points) The tool should support drawing a line segment.
Command: `line` *n x1 y1 x2 y2*
Effect: Creates a new line segment that has a name *n* and whose two ends are at locations (*x1*, *y1*) and (*x2*, *y2*), respectively.

[REQ4] (1.5 points) The tool should support drawing a circle.
Command: `circle` *n x y r*
Effect: Creates a new circle that has a name *n*, whose center is at location (*x*, *y*), and whose radius is *r*.

[REQ5] (1.5 points) The tool should support drawing a square.
Command: `square` *n x y l*
Effect: Creates a new square that has a name *n*, whose top-left corner is at location (*x*, *y*), and whose side length is *l*.

[REQ6] (2 points) The tool should support grouping a non-empty list of shapes into one shape.
Command: `group` *n n1 n2 ...*
Effect: Creates a new shape named *n* by grouping existing shapes named *n1*, *n2*, ... After this operation, shapes *n1*, *n2*, ... cannot be used directly in the following operations, until shape *n* is `ungroup`ed.

[REQ7] (2 points) The tool should support ungrouping a shape that was created by grouping shapes.
Command: `ungroup` *n*
Effect: Ungroups shape *n* into its component shapes. After this operation, shape *n* is not defined, but shapes *n1*, *n2*, ... can be used directly in the following operations.

[REQ8] (2 points) The tool should support deleting a shape.
Command: `delete` *n*
Effect: Deletes the shape named *n*. If a shape is a group, all its members are also deleted.

[REQ9] (4 points) The tool should support calculating the minimum bounding box[1] of a shape.
Command: `boundingbox` *n*
Effect: Calculates and outputs the minimum bounding box of the shape name *n* in the format "*x y w h*", where *x* and *y* are the x and y coordinates of the top-left corner of the bounding box, while *w* and *h* are the width and height of the bounding box. Note that the bounding box of a group shape contains all the shapes in the group. All floating point numbers should be rounded to 2 decimal places.

[REQ10] (4 points) The tool should support moving a shape.
Command: `move` *n dx dy*
Effect: Moves the shape named *n*, horizontally by *dx* and vertically by *dy*. If shape *n* is a group, all its component shapes are moved.

[REQ11] (4 points) The tool should support picking and moving a shape.
Command: `pick-and-move` *x y dx dy*
Effect: Picks the first shape that *contains* point (*x*, *y*) and moves it horizontally by *dx* and vertically by *dy*.

When drawing with the Clevis tool, shapes that are created later have larger Z-orders[2], and this operation checks the shapes in decreasing Z-order. A shape contains a point if and only if the minimum distance from the point to the *outline*, i.e., not including the inside, of the shape is smaller than 0.05. For example, a rectangle contains a point if and only if the minimum distance from the point to a side of the rectangle is smaller than 0.05. The outline of a group shape contains all the outlines of its component shapes.

[REQ12] (4 points) The tool should support reporting whether two shapes intersect with each other.
Command: `intersect` *n1 n2*
Effect: Reports whether two shapes *n1* and *n2* intersects with each other, i.e., whether their outlines share any common points.

[REQ13] (4 points) The tool should support listing the basic information about a shape.
Command: `list` *n*
Effect: Lists the basic information about the shape named *n*. For each simple shape, lists the types of information used to construct the shape. For instance, lists the *name*, the *center*, and the *radius* of a circle. For each group shape, lists the name of the group and all the shapes *directly* contained in the group. All floating point numbers should be rounded to 2 decimal places.

[1] https://en.wikipedia.org/wiki/Minimum_bounding_box
[2] https://en.wikipedia.org/wiki/Z-order

[REQ14] (2 points) The tool should support listing all shapes that have been drawn.
Command: `listAll`
Effect: Lists the basic information about all the the shapes that have been drawn in decreasing Z-order. Use indentation to indicate the containing relation between group shapes and their component shapes.

[REQ15] (2 points) The user should be able to terminate the current execution of Clevis.
Command: `quit`
Effect: Terminates the execution of Clevis.

The Clevis tool that you need to implement in this project will discard the created graphics when it is terminated. Support for operations like exporting the shapes to vector graphics formats can be added to Clevis in the future, but you do *not* need to take those operations into account when designing and implementing the system in this project. Having said that, you *do* need to use your best judgment to gracefully handle situations where a command is in bad format or a required action cannot be accomplished, e.g., because a name is already used or a name is not defined. Poor design in handling those situations will lead to point deductions.

The Clevis tool may be extended with the following *bonus* features:

[BON1] (8 points) Support for a graphical user interface (GUI) to demonstrate the resultant graphics after each drawing operation. If you implement this bonus feature, you may add a text field on the GUI for users to input the commands. Note that you may need to zoom in or out accordingly to provide a good enough view to the whole graphics. Also note that you still need to implement the command line interface even if your game has a GUI.

[BON2] (8 points) Support for the `undo` and `redo`[3] of all commands except `boundingbox`, `intersect`, `list`, `listAll`, and `quit`.

3 Group Forming

This is a group project. Each group may have 3 to 4 members. Please team up on Blackboard/Groups before or on October 4, 2021. Afterwards, we will randomly team up the ones with no group, and whoever requests for changing the group needs to

1. seek agreements from all the members in both his/her current group and the group that he/she wants to join, and
2. inform the instructor via email of the desired change, with agreements from the other members attached. The email should be cc-ed to all the relevant members. Upon receiving this email, we will make the corresponding change.

4 Code Inspection

The inspection tool of IntelliJ IDEA[4] checks your program to identify potential problems in the source code. We have prepared a set of rules to be used in grading (`COMP2021_PROJECT.xml` in the project material package). Import the rules into your IntelliJ IDEA IDE and check your implementation against the rules. Note that unit tests do not need to be checked against these rules.

The inspection tool is able to check for many potential problems, but only a few of them are considered errors by the provided rule set. 2 points will be deducted for each *error* in your code reported by the tool.

[3] https://en.wikipedia.org/wiki/Undo
[4] https://www.jetbrains.com/help/idea/code-inspection.html

5 Line Coverage of Unit Tests

Line coverage[5] is a measure used in software testing to report the percentage of source code lines that have been tested: The higher the coverage, the more thoroughly we have tested a program.

You should follow the Model-View-Controller (MVC) pattern[6] in designing CLEVIS. Put all Java classes for the model into package `hk.edu.polyu.comp.comp2021.CLEVIS.model` (see the structure in the sample project) and *write tests for the model classes.*

During grading, we will use the coverage tool of IntelliJ IDEA[7] to get the line coverage of your tests on package `hk.edu.polyu.comp.comp2021.CLEVIS.model`. Your will get 10 base points for a line coverage between 90% and 100%, 9 base points for a coverage between 80% and 89%, and so on. You will get 0 base points for a line coverage below 30%. The final points you get for line coverage of unit tests will be calculated as your base points multiplied by the percentage of your requirements coverage. For example, if you only implement 60% of the requirements and you achieve 95% line coverage in testing, then you will get 6 = 10 * 60% points for this part.

6 Design Review

The 3-hour lecture time in Week 13 will be devoted to the peer review of your design. The review will be conducted in clusters of X groups (X is to be decided); Each group needs to review the designs of the other groups in its cluster and fill out the review reports. In a review report, a group needs to point out which design choices from another group are good, which are questionable, which should have been avoided, and then explain the reasons.

Details about the organization of the design review and the format of the review report will be announced before the review session.

7 Project Grading

You can earn at most 100 points in total in the project from the following components:

- Requirements coverage (in total 40 points, as listed in Section 2)
- Code quality (10 points for good object-oriented design and 10 for good code style, as reported by the code inspection tool)
- Line coverage of unit tests (10 points)
- Presentation (7 points)
- Design review report (8 points)
- Final report (15 points)
- Bonus points (16 points)

  Note: Bonus points can be used to reach, but not exceed, 100 points.

In the project, each group member is expected to actively participate and make fair contributions. In general, members of a group will receive the same grade for what their group produces at the end. *Individual grading*, however, could be arranged if any member thinks “one grade for all” is unfair and files a request to the instructor in writing (e.g., via email). In individual grading, the grade received by each group member will be based on his/her own contributions to the project, and each member will need to provide evidence for his/her contributions to facilitate the grading.

[5]http://en.wikipedia.org/wiki/Code_coverage
[6]https://en.wikipedia.org/wiki/Model-view-controller
[7]https://www.jetbrains.com/help/idea/code-coverage.html

8 General Notes

- Java SE Development Kit Version 16 [8] or 17[9] and IntelliJ IDEA Community Edition Version 2021.2[10] will be used in grading your project. Make sure you use the same versions of tools for your development.
- Your code should not rely on any library that is not part of the standard Java SE 16 or 17 API.
- The project material package also include three other files:
  - `SampleProject.zip`: Provides an example for the organization of the project. Feel free to build Clevis based on the sample project.
  - `IntelliJ IDEA Tutorial.pdf`: Briefly explains how certain tasks relevant to the project can be accomplished in IntelliJ IDEA.
  - `COMP2021_PROJECT.xml`: Contains the rules to be used in code inspection.

If you use other IDEs for your development, make sure all your classes and tests are put into an IntelliJ IDEA project that is ready to be tested and executed. 50 points will be deducted from projects not satisfying this requirement or with compilation errors!

[8] https://www.oracle.com/java/technologies/javase/jdk16-archive-downloads.html
[9] https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html
[10] https://www.jetbrains.com/idea/download/other.html

9 Project Report Template

Project Report
Group XYZ
COMP2021 Object-Oriented Programming (Fall 2021)
Member1
Member2
Member3

1 Introduction

This document describes the design and implementation of the Clevis tool by group XYZ. The project is part of the course COMP2021 Object-Oriented Programming at PolyU.

2 A Command-Line Vector Graphics Software

In this section, we describe first the overall design of Clevis and then the implementation details of the requirements.

2.1 Design

Use class diagram(s) to give an overview of the system design and describe in general terms how different components fit together. Feel free to elaborate on design patterns used (except for the MVC pattern) and/or anything else that might help others understand your design.

2.2 Requirements

For each (compulsory and bonus) requirement, describe 1) whether it is implemented and, when yes, 2) how you implemented the requirement as well as 3) how you handled various error conditions.

[REQ1] 1) The requirement is implemented.
2) Implementation details.
3) Error conditions and how they are handled.

[REQ2] 1) The requirement is not implemented.

[REQ3] ...

3 User Manual

In this section, we explain how Clevis works from a user's perspective.

Describe here all the commands that the system supports. For each command, use screenshots to show the results under different inputs.