Group Project: A Command Line Vector Graphics Software

Department of Computing

18th September 2025

Deadlines

1. Form your groups of 3 to 4 students before 14:00, 2nd October 2025. Afterward, students with no group will be randomly grouped. Requests for group change will be approved only if written agreements from all members of the involved groups are provided before 14:00, 9th October 2025.
2. Compress the source code (including the tests), the tests, the user manual, the presentation slides and recording, and the project report into a single ZIP archive and submit it on Blackboard before 14:00, 24th November 2025.
3. Submit the design review reports in PDF or JPG format on Blackboard before 14:00, 27th November 2025.
4. Late submissions will lead to 25% deduction of points per day.

1 Overview

The goal of this project is to develop a Command LinE Vector graphIcs Software (Clevis) in Java. Users should be able to create and manipulate vector graphics containing shapes such as 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`, and all shapes have *unique* names. Note that all double values should be rounded to 2 decimal places when printed out.

[REQ1] (4 points) All of 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, e.g., `java hk.edu.polyu.comp.comp2021.clevis.Application -html d:\log.html -txt d:\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 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 operations defined by [REQ8] through [REQ13], 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 operations defined by [REQ8] through [REQ13].

[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 of the shapes in the group.

[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 finding the topmost shape that covers a point.
Command: `shapeAt` *x y*
Effect: Returns the name of the shape with the highest Z-index that covers point (*x*, *y*).

When drawing with the CLEVIS tool, shapes created later are on top of those created earlier and have higher Z-indices[2].

A shape *covers* 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,

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

a rectangle covers 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 of 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* intersect with each other, i.e., whether their minimum bounding boxes share any internal 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 of the shapes *directly* contained in the group.

[REQ14] (2 points) The tool should support listing all shapes that have been drawn.
Command: `listAll`
Effect: Lists the basic information about all of 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. That said, you *do* need to use your best judgment to gracefully handle situations where a command is ill-formed 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 view to the whole graphics. Also note that you still need to implement the command line interface even if your tool can run in the GUI mode.

[BON2] (4 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 from the same class session, i.e., *no group across different class sessions is allowed*. Form your groups before 14:00, 2nd October 2025. Afterward, students with no group will be randomly grouped. Requests for group change will be approved only if written agreements from all members of the involved groups are provided before 14:00, 9th October 2025.

[3] https://en.wikipedia.org/wiki/Undo

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. Unit tests do not need to be checked.

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. One point will be deducted for each *error* in your code reported by the tool.

5 Line Coverage of Unit Tests

The line coverage[5] achieved by a suite of unit tests is the percentage of source code lines that the tests exercise, and it is an important measure of the test suite's quality. The higher the coverage, the more thoroughly the suite of tests exercised a program.

You should follow the Model-View-Controller (MVC) pattern[6] in designing the CLEVIS tool. 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`. You will get 10 base points for line coverage between 90% and 100%, 9 base points for coverage between 80% and 89%, and so on. You will get 0 base points for 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, 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 later); Each group needs to review the design of every other group in its cluster and fill out a review report. That is, each group will need to fill out and submit X-1 review reports, one for every other group in its cluster. In each report, the reviewer group needs to point out which design choices from the reviewee group are good, which are questionable, which should have been avoided, and explain the reasons.

Details about the design review's organization and the report's format will be announced before the review.

7 Project Report

Each group must submit a project report that summarizes the group's work in the project, and you may refer to Section A for the required structure and contents for the report.

Individual contributions: In particular, your group needs to reach a consensus on the percentage of contributions each member has made to the project, and the percentages need to be clearly stated right below the report title. For example, if your group has 3 members and they have made equal contributions, you may write that each member has completed 33.3% of the work. Refer to Section 11 for how this percentage influences your individual score for the project.

[4] https://www.jetbrains.com/help/idea/code-inspection.html
[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

If your group has problems in reaching such a consensus, you should submit all of the other required materials before the deadline and report your case to the instructor before 14:00, 24th November 2025.

Learning-to-Learn: Some requirements in this project concern aspects of Java and/or object-oriented programming that are not fully covered in lectures, and you need to plan, conduct, evaluate, and adjust your self-learning activities to master the related knowledge and accomplish the corresponding tasks. Each group needs to 1) reflect on their learning-to-learn experiences in the project and 2) describe the plan to improve their self-learning approaches in a section named "Learning-to-Learn" of the final report.

8 Use of GenAI

Using GenAI tools in the project is allowed, but you should properly acknowledge all of the contents in your deliverables that were created with the help of those tools. More specifically, each group must fill out the "Honour Declaration for Group Project" and include it within the final submission, even if the group has not used GenAI in the project. You may refer to https://www.polyu.edu.hk/ar/docdrive/polyu-students/Student-guide-on-the-use-of-GenAI_revised_250214.pdf for more guidance on the use of GenAI.

9 Presentation

The presentation should not exceed 6 minutes, and each group member must present at least 1 minute.

Your presentation should at least include the following parts: 1) the overall architecture of your program, 2) a design choice for the program that concerns inheritance and polymorphism, and 3) a short discussion on how object-oriented design/programming helped you improve the reusability/scalability of the code. You may also include screen snapshots or short video clips of the CVFS running in your presentation, but do not let them dominate or excessively consume the overall presentation time.

Your presentation video should be in MP4 format. To facilitate validating the presenters, all presenters are required to show their student ID cards and faces at the beginning of the presentation, and the current presenter's face must always be shown in the video.[8] Each group member who does not show his/her face or student ID card in the presentation will have half of his/her marks deducted.

10 Submission

Each group is required to submit a single ZIP file containing the project code (including the source and test code), the user manual, the presentation slides and video, the project report, and the Honour Declaration for Group Project. Make sure that all of the documents you submit, unless specified otherwise, should be in PDF format. You should organize the files and directories in a consistent structure to ensure clarity and ease of access. Here are the guidelines for preparing the ZIP file:

1. Create a main directory with your group ID (and, optionally, your project's name or another suitable identifier).
2. Within the main directory, create two subdirectories for the ProjectCode and Presentation.
3. Place the relevant files in the required format in their corresponding directories. For example:
    - The project report should be placed inside the main directory.

[8] All presentation videos will be automatically destroyed in the system after one year to protect your privacy.

- The Honour Declaration for Group Project should be placed inside the main directory.
- The user manual should be placed inside the main directory.
- The IntelliJ IDEA project folder should be placed inside the ProjectCode directory.
- The presentation slides and video should be placed inside the Presentation directory.

4. Select all of the directories and files within the main directory.
5. Compress the selected items into a ZIP file.

Example Directory Structure:

```
GroupID
├── project_report.pdf
├── Honour Declaration for Group Project.pdf
├── user_manual.pdf
├── ProjectCode
│   └── The IntelliJ IDEA project folder
└── Presentation
    ├── slides.pdf
    └── video.mp4
```

11 Grading

The overall project work of your group can earn at most 100 points in total 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)
- Individual report and user manual (15 points)
- Bonus points (12 points)

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

Your individual score for the project will be calculated based on the score for the overall project work of your group and your percentage of contributions to the project. Suppose the overall project work of a group is worth x points, the group has y members ($3 \leq y \leq 4$), and one member's contribution is $z\%$. The member's individual score for the project is calculated as $\min(x * y * z\%, x)$. We encourage equal contribution.

Failure to submit the group project Honour Declaration, or submitting a false declaration, will result in a penalty of up to 40% of the total points.

12 General Notes

- Java SE Development Kit Version 21[9] and IntelliJ IDEA **Community Edition** (Version 2024.2[10]) will be used in grading your project. Make sure you use the same versions of tools in your development. Note that you need to configure "File | Project Structure... | Project Settings | Project" in the IntelliJ IDEA IDE as the following:

[9] https://www.oracle.com/java/technologies/javase/jdk21-archive-downloads.html
[10] https://www.jetbrains.com/idea/download/other.html

  - SDK: 21
  - Language level: 8 - Lambdas, type annotations etc.

- Your code should not rely on any library beyond the standard Java SE Development Kit 21 API.
- The project material package also includes four other files:
  - `Honour Declaration for Group Project.docx`: Allows you to declare your use of GenAI, if any, in the project.
  - `SampleProject.zip`: Provides an example for the structure of the project directory. Feel free to build your system 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 for code inspection.
- If you use other tools and/or IDEs to program your system, make sure all your classes and tests are put into an IntelliJ IDEA project that is ready to be compiled and executed. **50 points will be deducted** from a project not satisfying this requirement or with compilation errors!

If you use other tools and/or IDEs to program your system, make sure all your classes and tests are put into an IntelliJ IDEA project that is ready to be compiled and executed. **50 points will be deducted** from a project not satisfying this requirement or with compilation errors!

Appendixes

A Project Report Template

Project Report
COMP2021 Object-Oriented Programming (Fall 2025)
Group XYZ

Members and contribution percentages:
Member1: a%
Member2: b%
Member3: c%
Member4: d%

1 Introduction

This report outlines the design and implementation of the Command Line Vector Graphics Software (Clevis), a project completed by Group XYZ for the course *COMP2021 Object-Oriented Programming* at PolyU.

2 The Command Line Vector Graphics Software (Clevis)

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

2.1 Design

Use class diagram(s) to give an overview of the system design and explain how different components fit together. Feel free to elaborate on the design patterns employed and/or justify critical design decisions to facilitate a comprehensive understanding of the system's construction.

2.2 Implementation of 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 error handling.

[REQ2] 1) The requirement is not implemented.

[REQ3] ...

3 Learning-to-Learn

Use this section to 1) reflect on your learning-to-learn experiences in the project and 2) describe your plan to improve your self-learning approaches.

B User Manual

To prepare a user manual, it's important to note that there is no one-size-fits-all template. User manuals can differ significantly based on the system being documented and the specific needs of the users. However, the following guidelines can help you create an effective user manual:

- Introduction: Begin by providing an overview of the system and its purpose. Explain how the system works from a user's perspective.
- Commands Description: Describe all of the commands that the system supports. Provide detailed explanations of each command, including their functionalities and how they can be used.
- Screenshots: For each command, consider including screenshots to visually demonstrate the results under different inputs. Capture screenshots that showcase the system's response or output when specific commands are executed. Include captions or annotations to provide additional context or explanations for each screenshot.
- Step-by-Step Instructions: Provide clear and concise step-by-step instructions for users to follow when using each command. Break down the process into logical steps, ensuring that users can easily understand and replicate the actions.
- Troubleshooting: Include a troubleshooting section to address common issues or errors that users may encounter while using the system. Provide solutions or workarounds for each problem, ensuring that users can resolve issues on their own.
- Additional Resources: If applicable, include additional resources such as references, FAQs, or contact information for technical support. Provide users with avenues to seek assistance or further information if needed.
- Formatting and Organization: Ensure that the user manual is well-formatted and organized. Use headings, subheadings, and bullet points to improve readability. Consider using a consistent and intuitive structure throughout the manual.

Remember, the purpose of a user manual is to provide clear instructions and support to users. Tailor the manual to meet the specific needs of your system and its users.