# Project Report

## COMP2021 Object-Oriented Programming

**Group 12**

Alibek KUANTKHAN GAO Zhilin
QI Shi Hao Zhejun HE

November, 2021

## 1 Introduction

**The CLEVIS tool** enables users “to create and manipulate vector graphics” of several types of shapes. The **input–process–output (IPO) model** is used to describe the requirements.

| Input | Process (Conception) | Output |
|---|---|---|
| 1. **Texts on the Command Line Interface** (the file name of HTML and TXT file, the details of a shape, other commands)<br><br>2. **Zoom in and out function** | **1. Get the file name from the input to create HTML and TXT files, with the ability to cope with unexpected errors during the input process (Robustness).**<br><br>**2. Get the details of a shape from the input, including its property and its parameters, and then collect it in a container that could contain different types of shapes.**<br><br>**3. Able to convert the message (parameters) of different shapes in the container into the real shape on the GUI.**<br><br>**4. Utilize Scroll Pane and Slider to enable zoom in/out operation.**<br><br>**5. Utilize stack to facilitate undo/redo operation.** | 1. **An HTML file and a TXT file** containing the message of input.<br><br>2. **A graphical user interface** that could refresh itself immediately when the user input some new commands on the command line. The program adjusts the rough scale automatically so that graphs of different sizes and locations fix in the GUI window. |

# 2 Command LinE Vector graphIcs Software (CLEVIS)

## 2.1 Design

### Design of HTML and TXT files

The HTML and TXT files are the core part of the application. The files' names are inputted as application parameters at the launch. If the specified files are not inputted, the program terminates. If the specified files already exist, then the program throws an IO exception. In order to realize the file read/write, it was decided to use FILE class in the java standard library. It allows the application to open, read and write various files. Both HTML and TXT files are represented by two different fields in the "FileCreator" class. When the class receives a successful command, the command is written by a "FileWriter". The TXT file uses the "write(string)" function. The HTML file already contains the initial code to create a basic HTML table. After the command is executed, it is covered by a HTML code and is written to a file. Every time the HTML file is written, the variable called "index" is incremented to keep a track of numbers of commands executed. Only executed commands are written to the files.

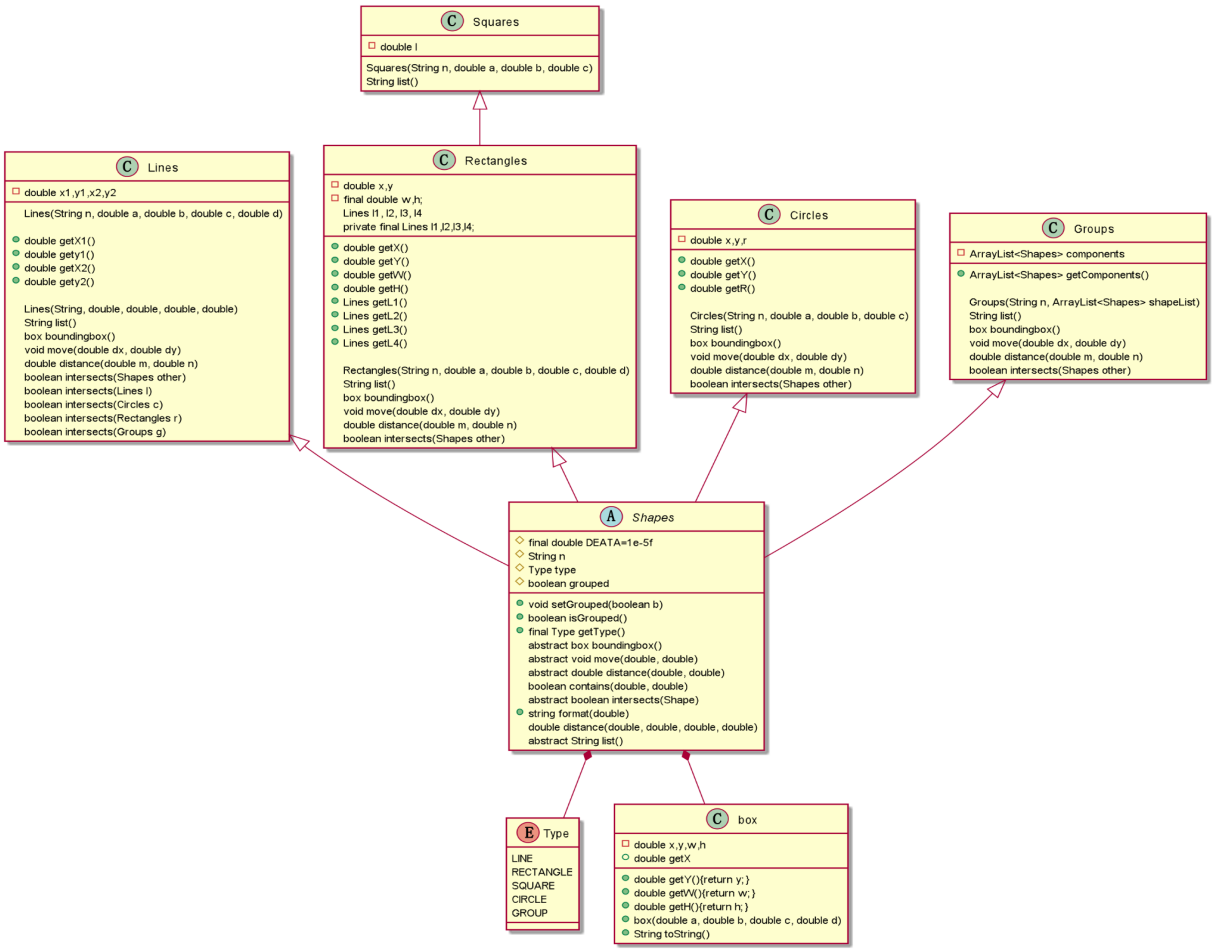
### Design of Shapes

Basic/fundamental part:
In order to implement requirements 2-13, a class named Shapes is defined first. Then several abstract methods are deployed to customize the code structure.

```
abstract String list();//REQ13
abstract box boundingbox();//REQ9
abstract void move(float dx, float dy);//REQ10
//cauculates the distance to a point
abstract float distance(float dx, float dy);
abstract boolean intersects(Shapes other);//REQ12
```

Due to this structure, every shape class contains the methods "list", "bounding box", "move", "distance", and "intersects".

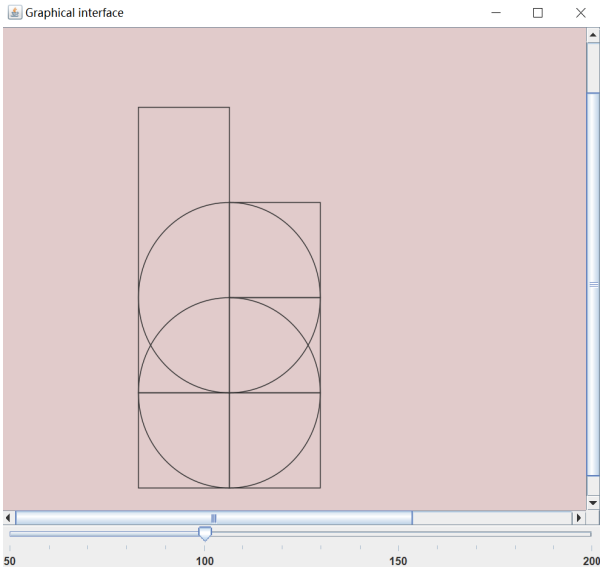
**Abstract class Shapes and inheritance**
To better illustrate the design idea, **the UML Class Diagram** is used to graphically represent the names, attributes and operations of each class, as well as the relationship of different classes, including abstract Shapes class, its subclasses, and others.




Shapes is a class specially used to process graphical data, and the structure must be easy to understand. Simultaneously, it is necessary to correctly use mathematical knowledge and consider the relationship between multiple graphics.

## Design of GUI

In the bonus part, the Swing is utilized, an interface providing GUI for Java programs to fulfill our goals. Moreover, we thought deeper to improve the usability of our program.




### Overview (Class MyWindow)

There are three classes, which are **MyWindow**, **MyFrame** and **MyPanel**. In the class **MyFrame** and **MyPanel**, JFrame and JPanel are utilized basically and if more functions are added, for example, JSlider, and modify the methods for better achieving the goals. In the class

**MyWindow**, MyFrame and MyPanel are instantiated, and we add JScrollPane into the instance frame to enable view shifting.

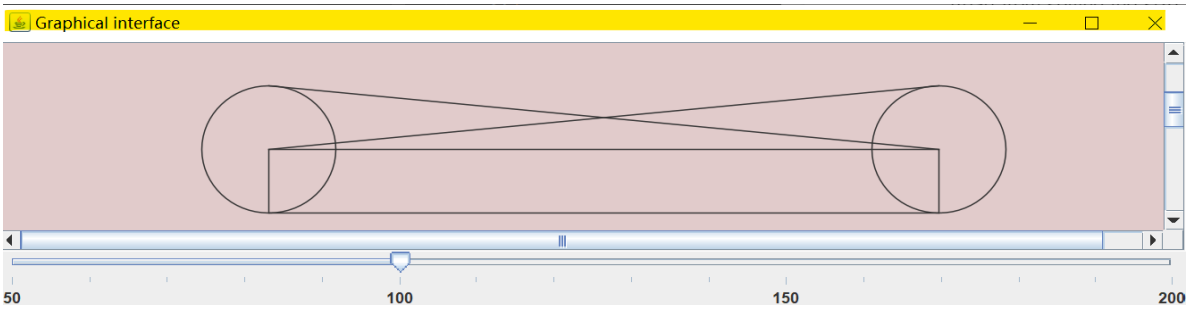
## Class MyFrame (Based on JFrame)


Apart from setting the size and the title of the frame, an operation the program will execute can be set when the user clicks the close button on the frame.

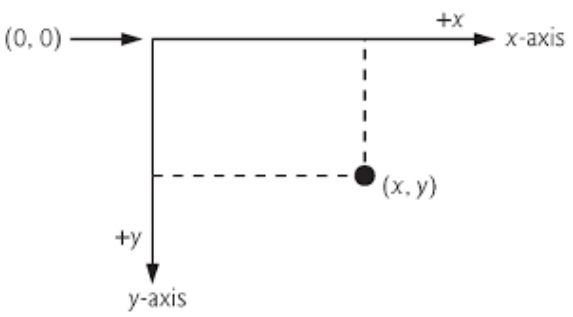
```
frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
```

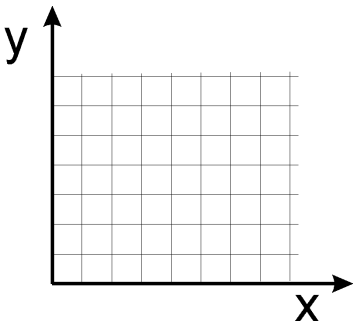
The reason why DISPOSE_ON_CLOSE is chosen lies in that there is no need to terminate the program when users click the close button on the frame; and the users would like to continue inputting something on the command line, and see the change of the shape later.

## Class Jpanel (Based on JPanel):




### (1) Modification of the Coordinate System in plain JPanel



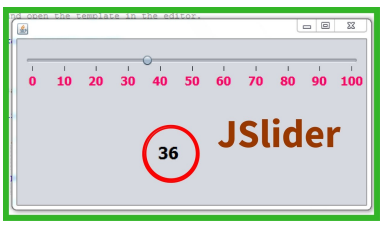




The Coordinate System in JPanel is different from the normal Cartesian coordinate system. Even though the top-left corner of the shapes (except for circle) is at location (x, y), we want to make the movement as normal as what in daily lives, instead of confusing the users. It is modified according to the calculations.

### (2) Add JSlider to enable zoom in/out

Rather than users' entering commands to enable zoom in/out, there is a better way to implement it: the slider. The method used is below.

```
final JSlider getControl(){...}
```



### (3) Set scales to enable preferred size for the viewer

```
Shapes.box b=g.boundingbox();
CommandLine.ungroup(g);
final int SCALEX=600,SCALEY=400,DELTAX=150,DELTAY=100;
double scale= Math.min(SCALEX / b.getW(), SCALEY / b.getH());
double dx=b.getX()*scale-DELTAX,dy=b.getY()*scale-DELTAY;

public final void stateChanged(ChangeEvent e){
  ...// if the state has changed, say the user just inputs a new shape
  scale=value/SCALAR;
  repaint(); // repaint its component.
  revalidate();}
```

Scale, dx and dy are three parameters that are set to help facilitate the modification.

The getX and getW methods are used in the Class **box**. Apart from that, the application utilizes Dimension to encapsulate the width and height of a component in the case of converting Shapes (defined class) to Shape, which is introduced below.

### (4) Difference of Shape and Shapes

```
private final ArrayList<Shapes> shapesList;
private Shape[] shapesList2;
```

**Shapes** is the abstract class to be defined, while **Shape** (i.e. the java.awt.Shape) is one of the most fundamental abstractions in Java 2D. Even though the fields and methods defined in the class Shapes could meet most of the requirements, it is found to be shallow to convert **Shapes** into **Shape**.

Thus the private **void initShapes()** converts all the elements of the type Shapes into the elements of the type Shape, which are stored in **shapesList2.** Then the graphics are printed out with precise parameters in the panel using a very concise method in Griphics2D.

```
Graphics2D g2 = (Graphics2D)g;
for (Shape s : shapesList2) {
  if (s == null) break;
  g2.draw(at.createTransformedShape(s));
}
```

*Note: The AffineTransform class represents a 2D affine transform that performs a linear mapping from 2D coordinates to other 2D coordinates that preserves the "straightness" and "parallelism" of lines. Affine transformations can be constructed using sequences of translations, scales, flips, rotations, and shears.*

## 2.2 Requirements

[REQ1]

1) The requirement is **implemented**.

2) Once the program receives parameters that contain log file names, it creates both files. A command is written to the files only when it is executed, so incorrect commands will not be logged.

3) If the files specified by the user already exist, or for some other reason the files cannot be created, then the program notifies the user and terminates.

[REQ2]

1) The requirement is **implemented**.

2) The program stores the coordinates for the top-left corner, height and width as fields. A rectangle consists of four lines, so in the constructor of class Rectangles, the logic is to build four lines based on the left top point (x, y): l1 represents the upper side, l2 represents the right side, l3 represents the downside and l4 represents the left side.

As it was designed in abstract class Shapes, it overrides methods like move, boundingbox, distance, and intersections. Methods “distance” and “intersects” are both based on class lines.

3) The user is notified to re-type if the number of parameters is incorrect or the name is invalid.

[REQ3]

1) The requirement is **implemented**

2) A line is represented by a Lines object. Because all shapes except a circle consist of lines, our computational methods are more focused on implementing and designing class “Lines”.

Core methods in class Lines:

distance:

Function distance plays an important role in testing intersection. It helps to calculate the distance between shapes and a single point, and its main idea is by comparing the distance between the point (m, n) and lines composed of (x1, y1) and (x2, y2).

It divides the problem into 3 general cases:

(1) When the line is perpendicular to the x-axis (2) when the line is perpendicular to the y-axis

(3) When the line is slant

Intersects:

The method “intersects(Lines l)” uses mathematical calculation to test whether two lines intersect each other, including the following cases:

(1) two lines perpendicular to the x-axis (2) two lines perpendicular to the y-axis

(3) an x-vertical line and a y-vertical line (4) a slant line and an axis-vertical line

(5) two slant lines

Other “intersects” methods are mainly based on the function "interacts(Lines l)".

[REQ4]

1) The requirement is **implemented**

2) The program stores the basic information and implements specific methods for circle in the class “Circles”.

3) In the beginning,concentric shapes were not considered, yet the issues arose when executing the test files. Hence, a condition was set to compare the distance - r with negative DELTA to solve this.

[REQ5]

1) The requirement is **implemented**

2) The definition of Square in math is “a special type of rectangle.” So, the class square directly extends class Rectangle, and the only difference is the “w=h” and “add a field” l.

[REQ6]

1) The requirement is **implemented**

2) In the class Groups, an ArrayList is created to store all the shapes, and there is a variable grouped for each shape to store its status. Once grouped equals to true, a shape will become inaccessible via direct operations on it.

3)The group can not be empty. Commands attempting this will not pass the checkArgs() test that checks the number of arguments in a line.

[REQ7]

1) The requirement is **implemented**

2) In this method the grouped status of a shape will be assigned back to

false first, then use remove() to delete the group.

3) Because the class shape is the parent of class Group, in practice it could contain both Groups and other shapes, while the group it contains may also contain some other shapes. Therefore, operations related to Groups are usually implemented recursively.

[REQ8]

1) The requirement is **implemented**

2) The utility of class delete and ungroup is a bit similar, but class delete obliterates all shapes in the target Group whereas ungroup() only eliminates a group type and changes the status of shapes in the target Group.

3) This method requires the shape reference, so it never tries to delete something that does not exist.

[REQ9]

1) The requirement is **implemented**

2) For shapes like squares and rectangles the program just needs to return its four basic fields. For lines, groups and circles, the method finds the bounding box mathematically.

[REQ10]

1) The requirement is implemented

2) Implementing method move() is easy, but considering the requirement in REQ6, various conditions were set in the command line. If the attribute “grouped” is equal to true, the program would ignore the command.

[REQ11]

1) The requirement is **implemented**

2) The graphics processing file (Shapes.java) only contains method move() to find the shape with the largest Z-order at run time by iterating the arraylist in which all shapes. are stored.

3) If no shape contains the parameter point, the user is notifed.

[REQ12]

1) The requirement is **implemented**

2) as mentioned in the requirement, if there aren’t any common points on their outlines, no interactions would be reported. Because of the definition of structure in class shape, every graphic class will have an interaction method. Among these methods, except for interacting with circles, all other shapes are based on interacts(Lines l).

[REQ13]

1) The requirement is **implemented**

2) to better represents the attributes of a shape, the input is as follows:

This is an example of the output of circles' list:

Name: "shape name"

Type: "shape type"

Centre: (x, y)

Radius: r

3) A problem occurred when rounding to 2 decimal places. Format() method is used to normalize it.

[REQ14]

1) The requirement is **implemented**

2) The program traverses the shapes list and list every shape that is not grouped. To generate correct indentations for shapes in some group, we call the listDetails() method recursively.

[REQ15]

1) The requirement is **implemented**

2) When the user types "quit", the application closes the graphiccal interface and terminates the whole program.

[BON1]

1) The requirement is **implemented.**

2) The design of the graphical user interface has been introduced in detail in 2.1.

[BON2]

1)The requirement is **implemented.**

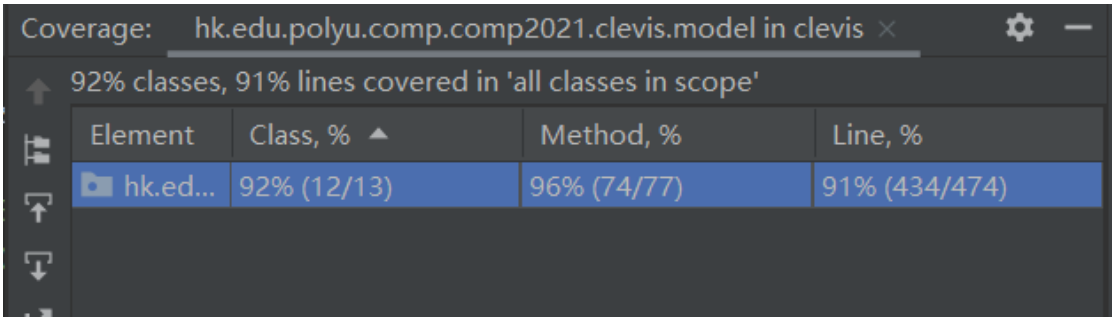
2) undo: After each modification to the shapes list,

redo: Once receive "redo", the program will run the method "Execute" again, but it won't clear the order has been input before.

3) The biggest problem part in this part is how we design "undo" well. There are so many conditions that could happen when the program is running. After we finish the framework of "undo", we input different orders many times only to implement "undo" and finish possible bugs. Now it is

good enough to run and test.

Line coverage:

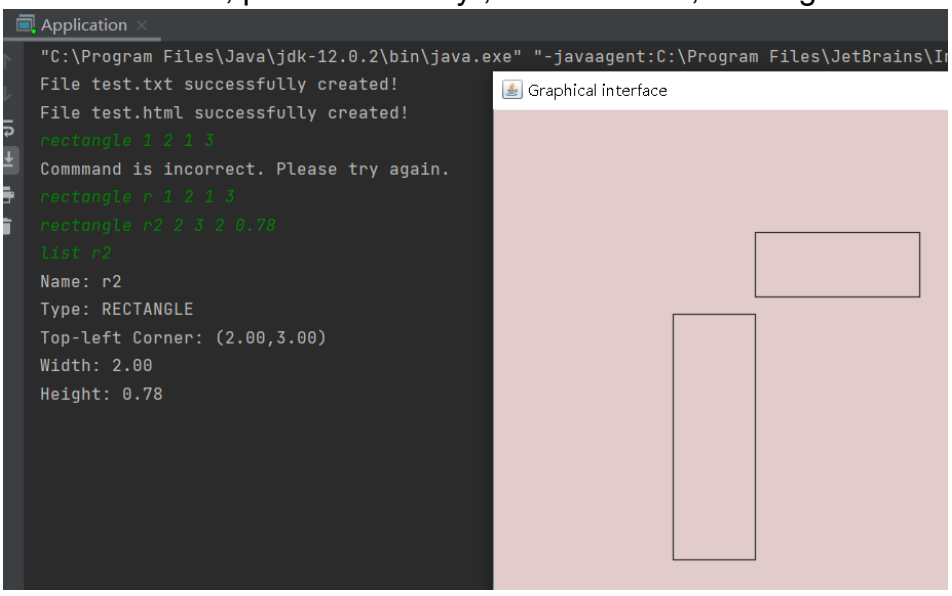



# 3 User Manual

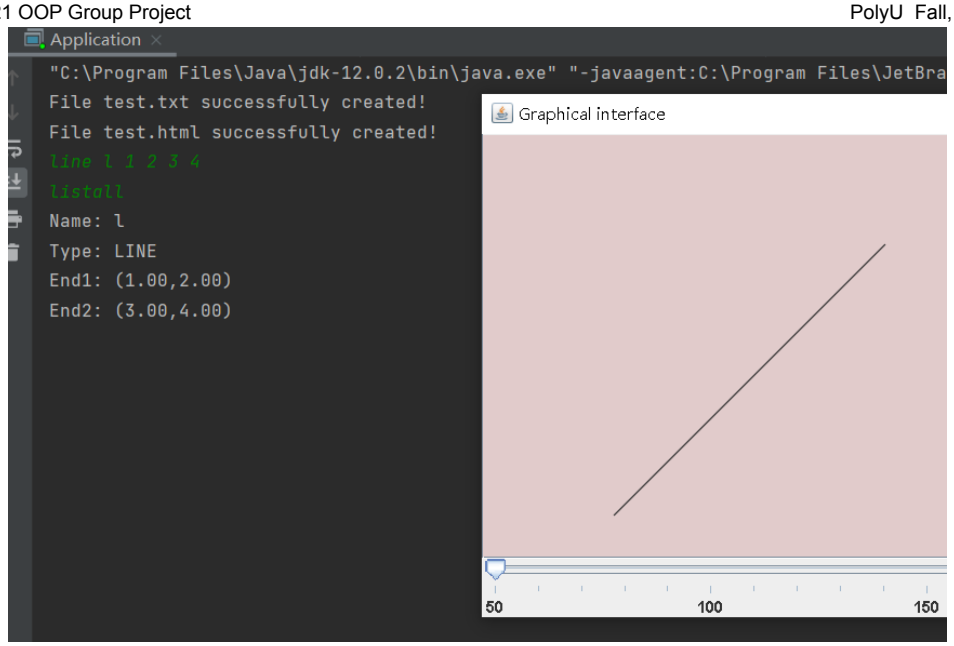
CLEVIS is a command line application that allows users to create, group, delete and inspect shapes. It encompasses 13 executable commands with unique functions and parameters. There are five types of commands: creational, removal, manipulative, informative and terminative.

**Creational commands**

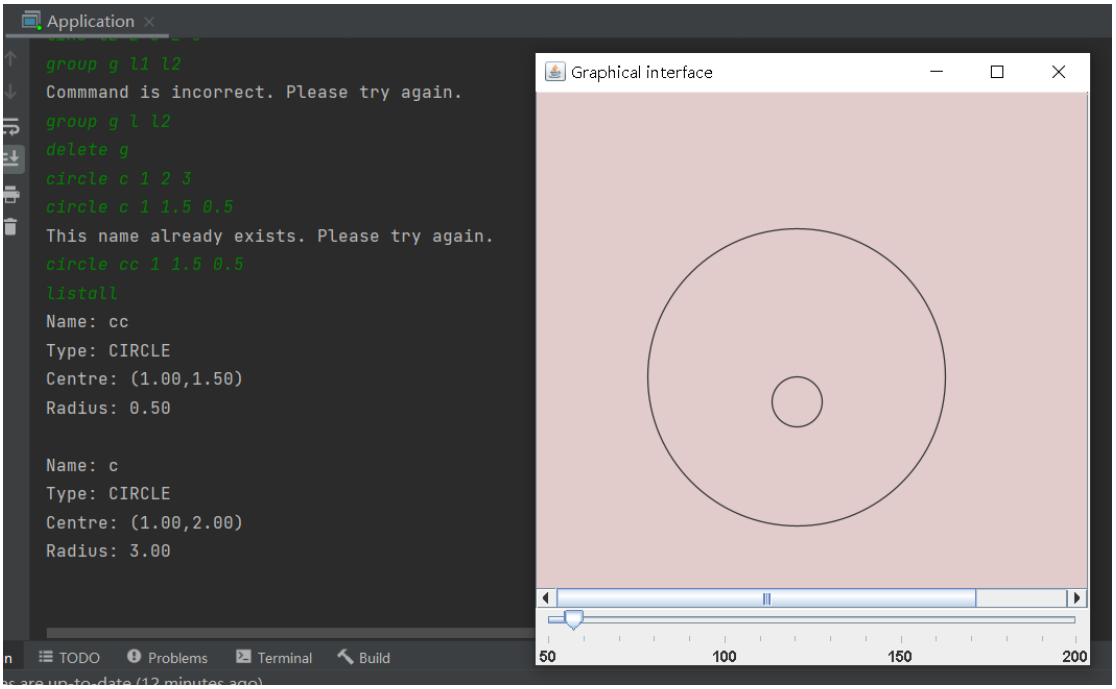
- “rectangle n x y w h” is a command that creates a new rectangle shape with the name n, points “x” and “y”, and “w” width, “h” height.



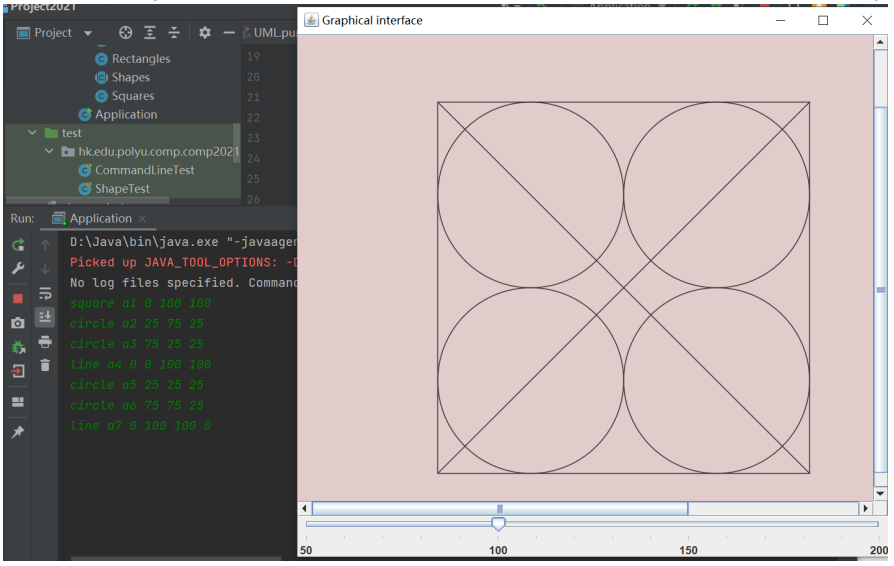

- “line n x1 y1 x2 y2” is a command that creates a new line with a name “n” from point (“x1”, ”y1”) to point (”x2”, “y2”).




- “circle n x y r” is a command that creates a new circle with a name “n” on a point (“x”, “y”) with a radius “r”.




- “square n x y l” is a command that creates a new square with a name “n” on a point (“x”, “y”) with sides of length “l”.

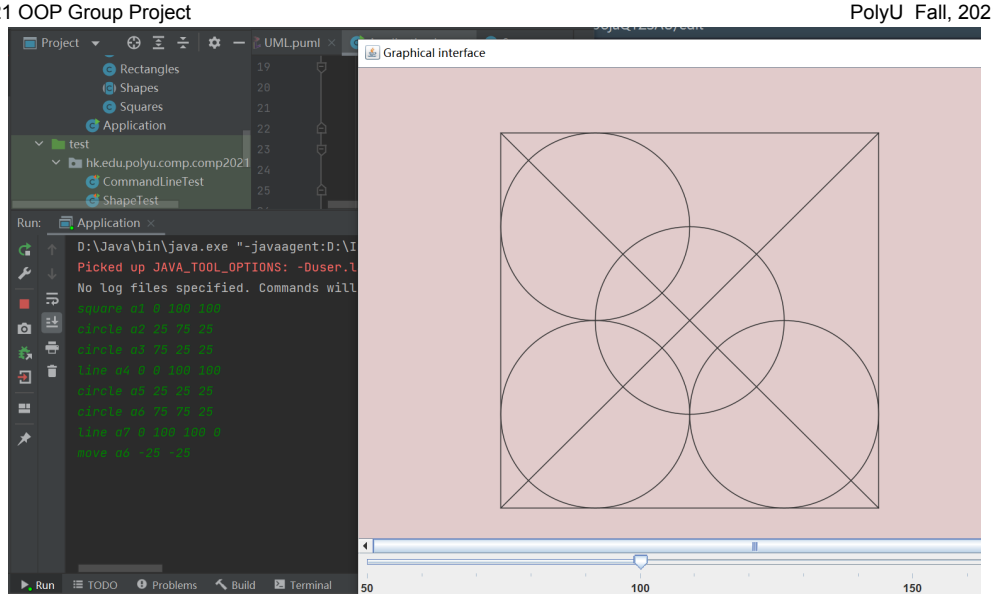



**Manipulative commands**

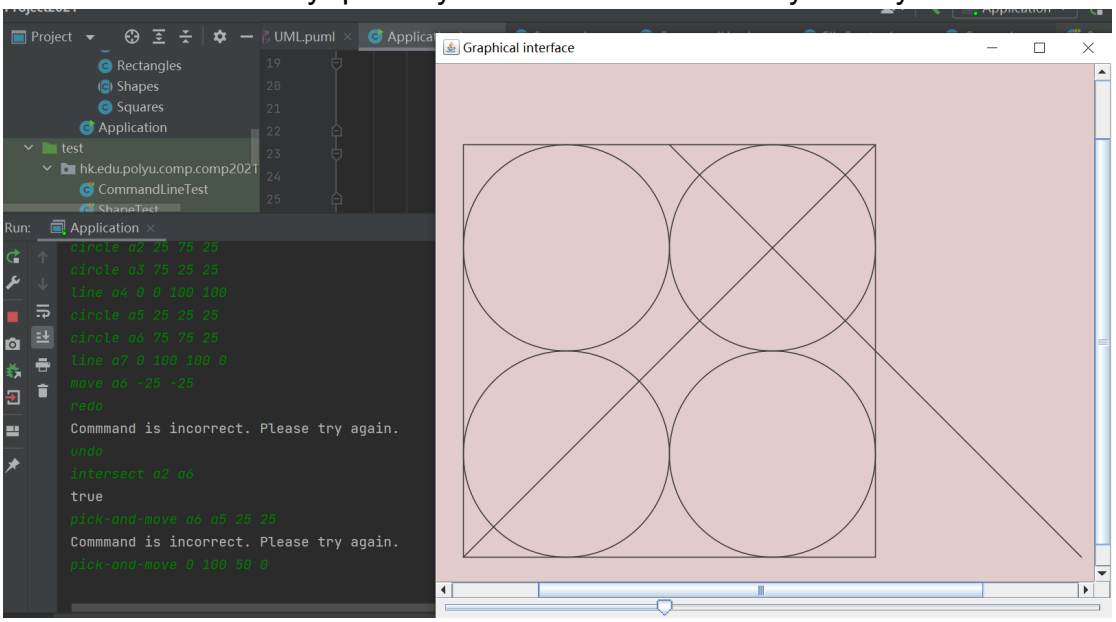
- “group n n1 n2...” is a command that creates a new group with a name “n” by grouping the specified shapes. Notice that once a shape was grouped, it can no longer be accessed through its name, only through the group.
- “ungroup n” is a command that ungroups the group with a name “n” and makes the shapes in it accessible. The ungrouped group will be undefined and unusable

```
list  g2
Name: g2
Type: GROUP
Components: g  c
list g
Shape does not exist.
ungroup g2
list g
Name: g
Type: GROUP
Components: r  s
```

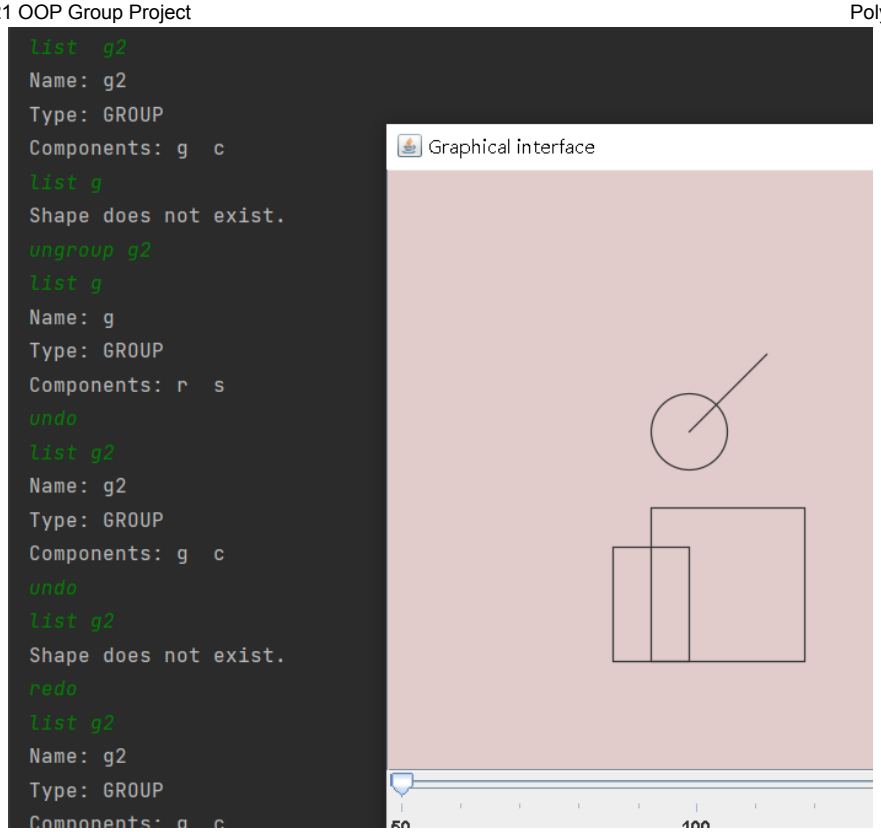
- “move n dx dy” is a command that moves the shape with a name “n” by “dx” in an “x” axis and “dy” in a “y” axis. The shape’s information is updated accordingly.




- “pick-and_move x y dx dy” is a command that moves the shape that first contains “x” and “y” points y “dx” in a “x” axis and “dy” in a “y” axis.

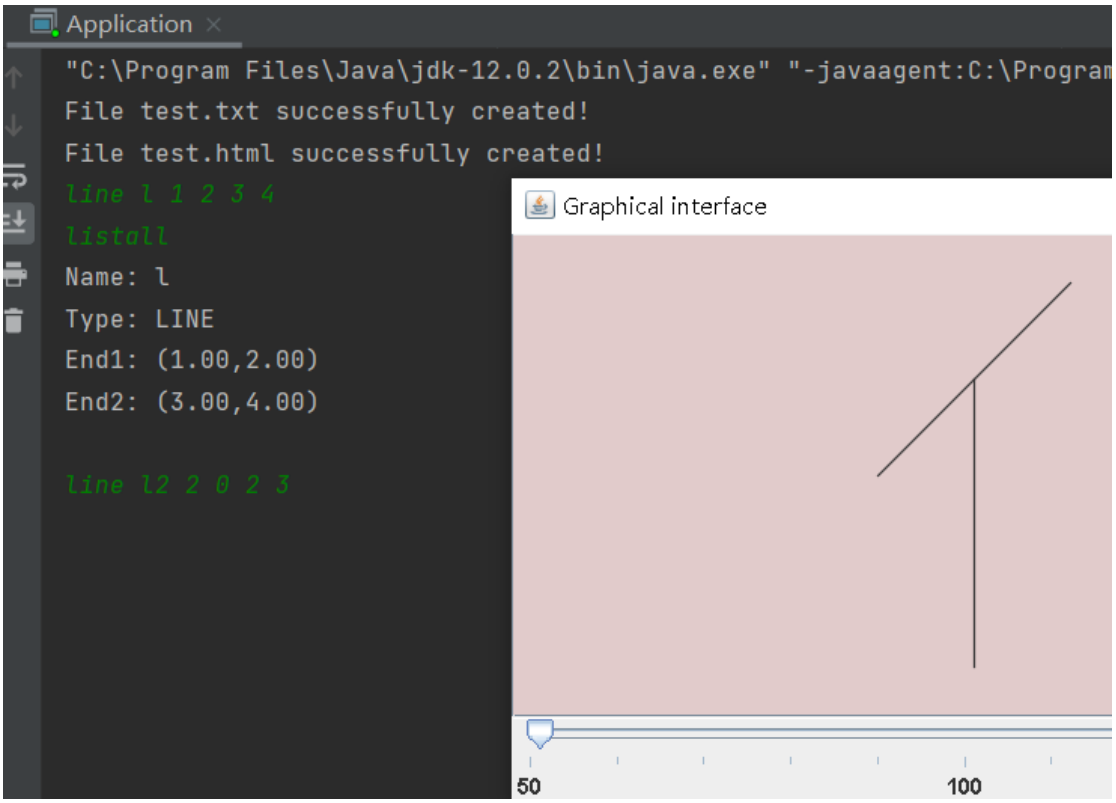


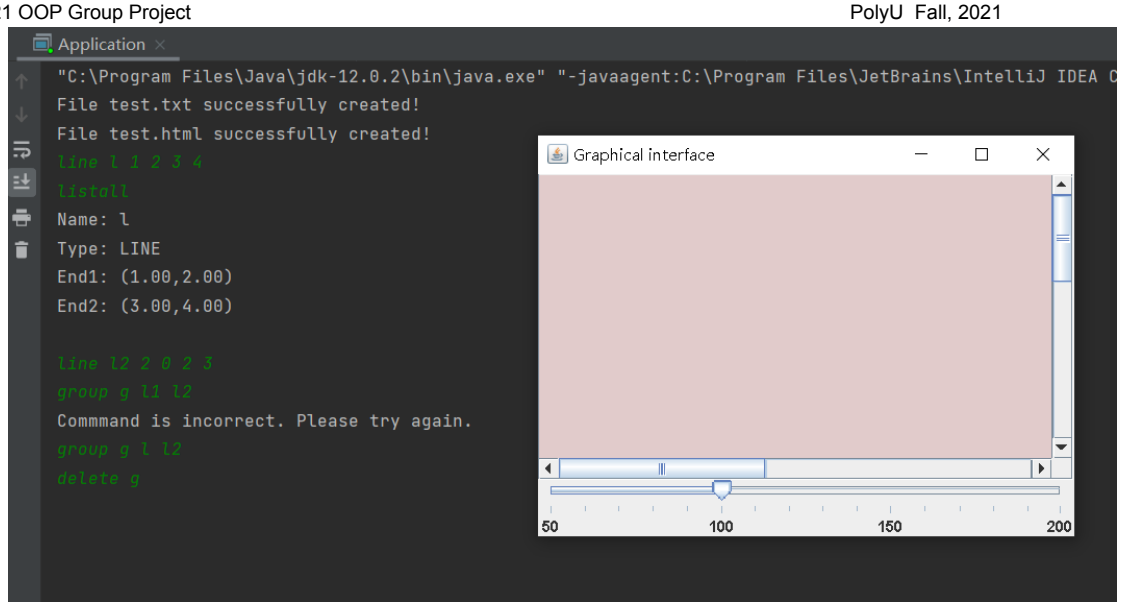

- “undo” and “redo”




**Removal commands**

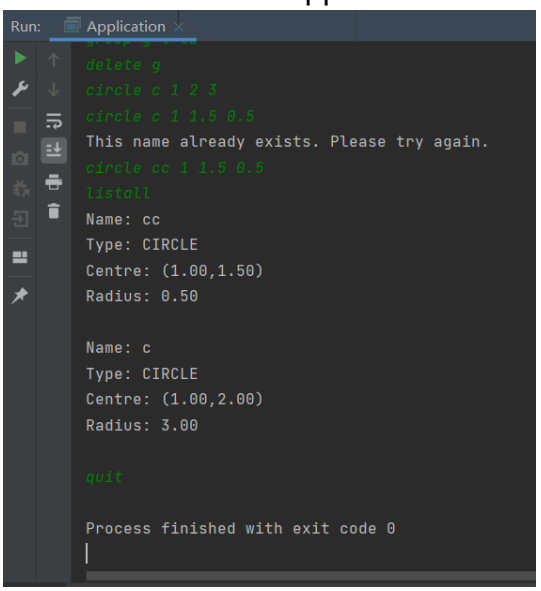
- "delete n" is a command that deletes the entity with a name "n". If it is a shape, it is deleted. If it is a group, the group itself and shapes it contains are deleted too.





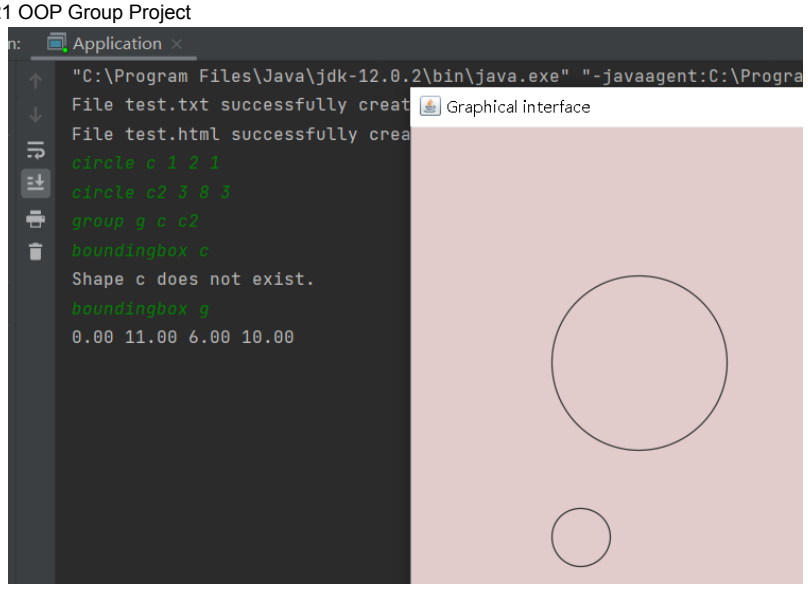

## Terminative commands

- "quit" is the only command without arguments. Its function is to terminate the whole application.

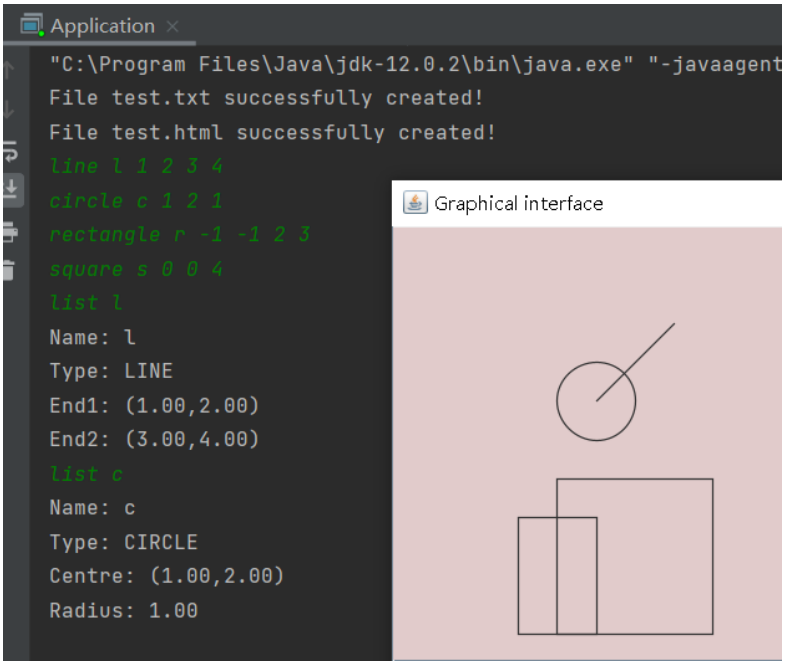


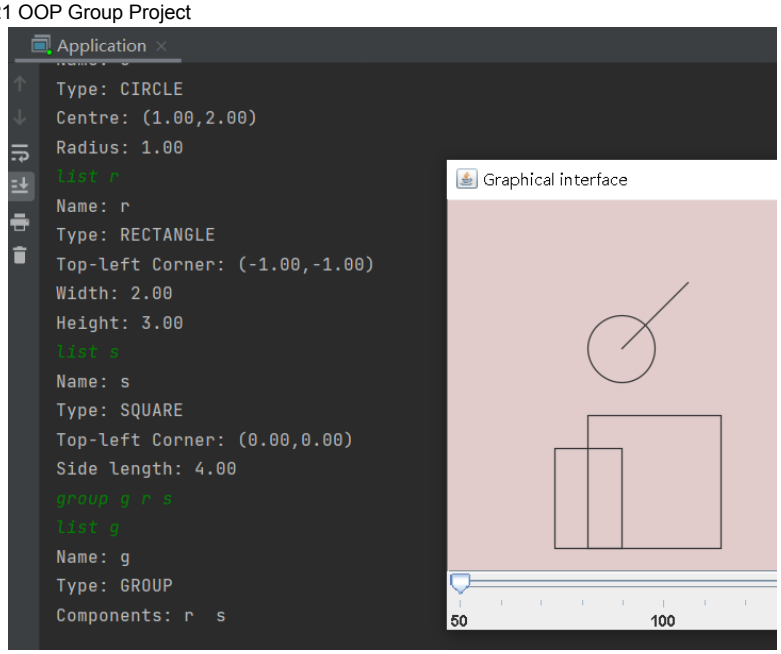

## Informative commands

- "boundingbox n" is a command that outputs the minimum bounding box of the shape named "n" in the format of "x y w h", where "x" and "y" are a point and "w" and "h" are width and height respectively.

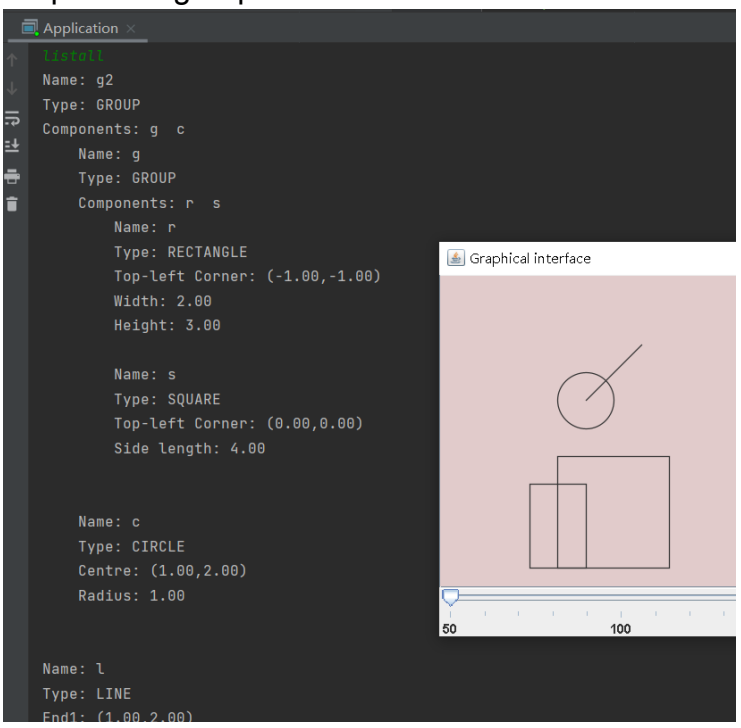



- “list n” is a command that outputs the information about the shape with a name “n”. The function does not work on grouped shapes.

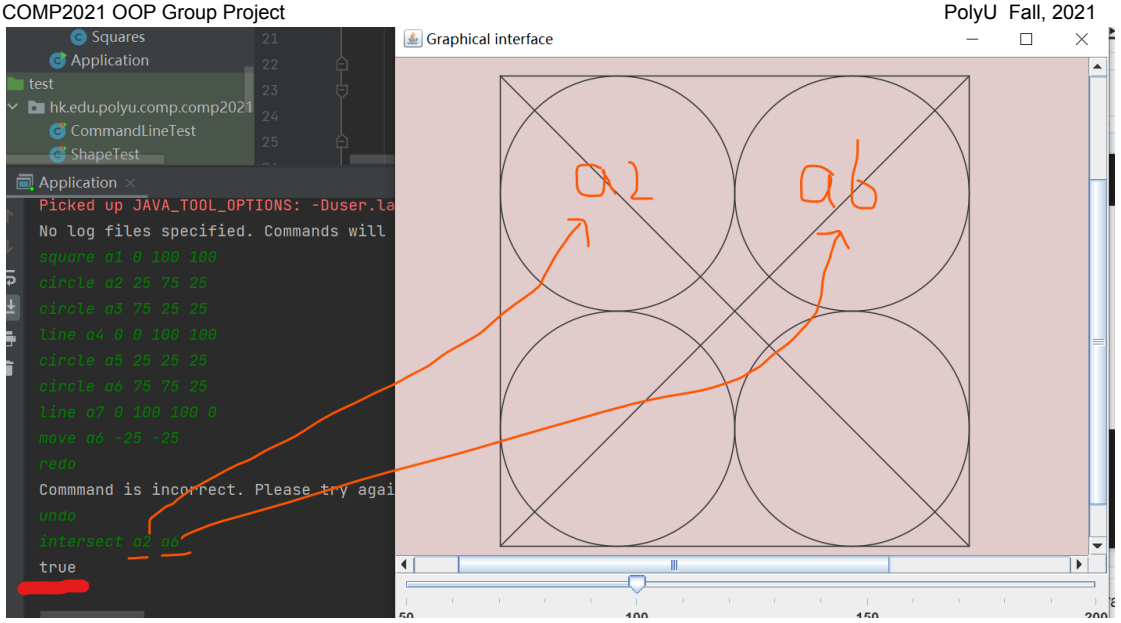





- “listAll” is a command that outputs the information about all of the shapes and groups.




- “intersect n1 n2” is a command that outputs “true” if two shapes named “n1” and “n2” do intersect, and false otherwise.




## Usage process

Note: After you input the file name in “Edit Configuration” -> the second line in “Build and run” as the format in the picture. You can start running the file “Clevis”.

But please remember, every time before restarting the program, please delete the two files “a.html” and “a.txt” or change the files’ name in the line. E.g:
-html b.html -txt b.txt


1. When the application is launched, the user needs to pass the names of the “.txt” and “.html” files as parameters for the commands to be logged. In case if such a file with the same name exists, the application terminates. When the file is created, the user may start using the functionality of the CLEVIS. Every successful command is recorded in the log files, until the program terminates.

2. After the input of the files name is done, the application is ready to get the commands. What is needed from a user is to input the command. **If the command is not correct, the application notifies the user, and the GUI of the graphics appears.** For example, if instead of the “rectangle n x y h w” user inputs “ectangle n x y h w”, the application will notify one that the specified command is not correct. **However, the user can input the commands with excessive spaces, in lower/upper cases interchangeably and CLEVIS will recognize it (Not case-sensitive).** For example, if user inputs “SqUarE new_square 23 2 1 4” the command will work and will be saved in an appropriate manner. If the user inputs lexically

correct but logically incorrect command, the application will notify one. For example, if the input is “group shapes square1 square 2 circle2 ”, but one of the shapes does not exist or was already grouped, the application notifies the user.

3. **If the user decides to exit the program, he can input the “quit” command** (**Closing the frame does not work)** and the program will be terminated. All of the input commands will be saved in the “.html” and “.txt” files.