Antonio Skondras

763787

Tietotekniikka (Computer Science)

Year of studies: 2021

27.4.2022

151 Drawing Program

General description

I created a simplistic drawing application which can draw different objects from rectangles to circles. The application has all necessary methods and classes that moderate project demands, for example users can draw different shaped and colored objects onto the image or if they don't want to use any shapes they simply can draw with a pen.

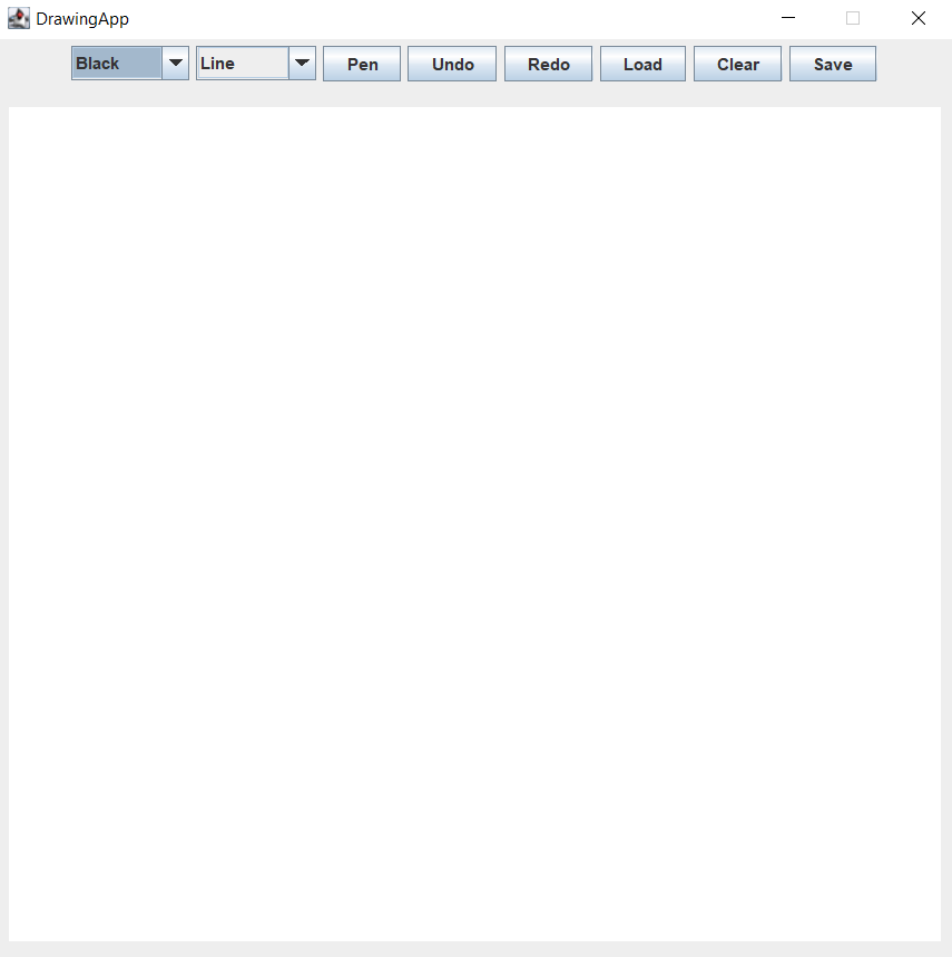
Also, users can undo or redo their sketching if they aren't satisfied with the result. Clearing the whole sketching is also possible with click of a button.

Saving and loading pictures in text-based format into a file is also possible. You can choose a file where you want to store your picture's information and load it from there afterwards.

At first, I wanted to do my project in a demanding difficulty level, but time went short, so I had to switch to moderate. I still did some extra features that moderate difficulty level didn't demand, for example redo –method and drawing with a pen feature.

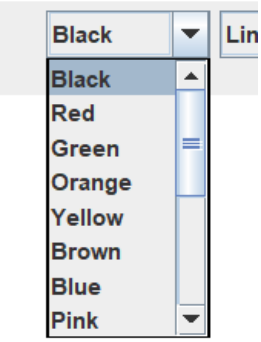
User interface

Starting the program is possible by going to the GUI object and pressing the play button. After that a new window of the program pops up and it looks like this.



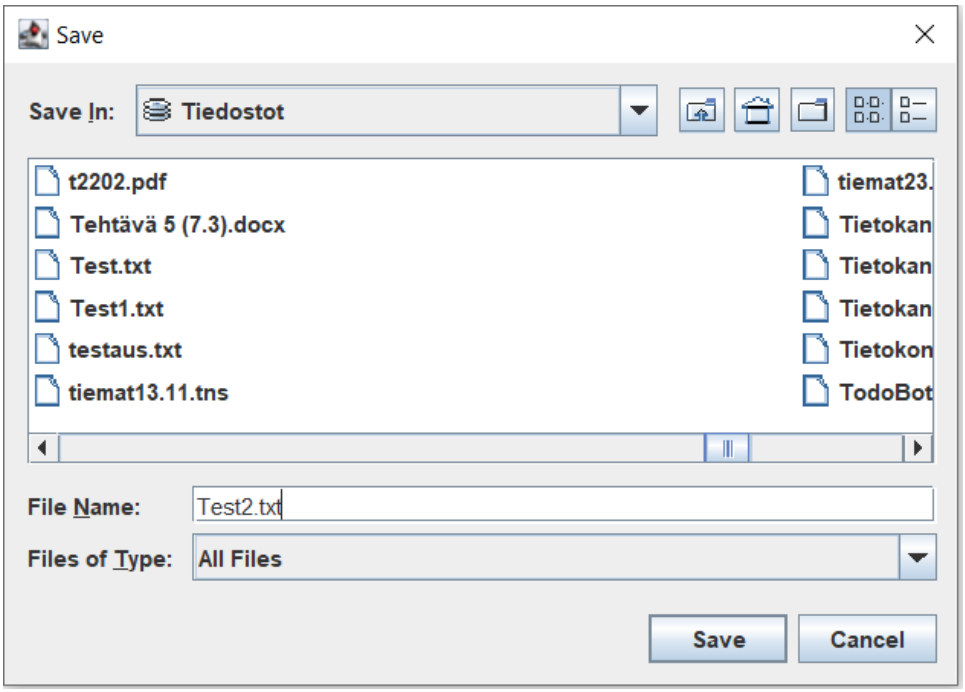
As you can see the program has two drop-down menus for the colors and the shapes.

The default color and shape are black and line. Choosing another color or shape is possible by clicking the drop-down menu. When clicking one of the catalogs you can see the options that the program has to offer and, in this example, different color options. All in all, there are 16 different options for color choosing and four different options for shapes. Search different options by dragging the bar.



After you are done with the settings you can start drawing by clicking the drawing area with the mouse and dragging it. During the drawing you can switch to different color, shape or undo your drawing by clicking the undo –button. Clicking the undo-button removes recent changes in your drawing and stores it in case you want to bring it back. If you accidentally undid something, you could click redo-button and it restores recent changes. You can clear the whole drawing as well by clicking the clear- button but keep in mind that after clicking the clear button you can't restore the drawing if you didn't save it beforehand.

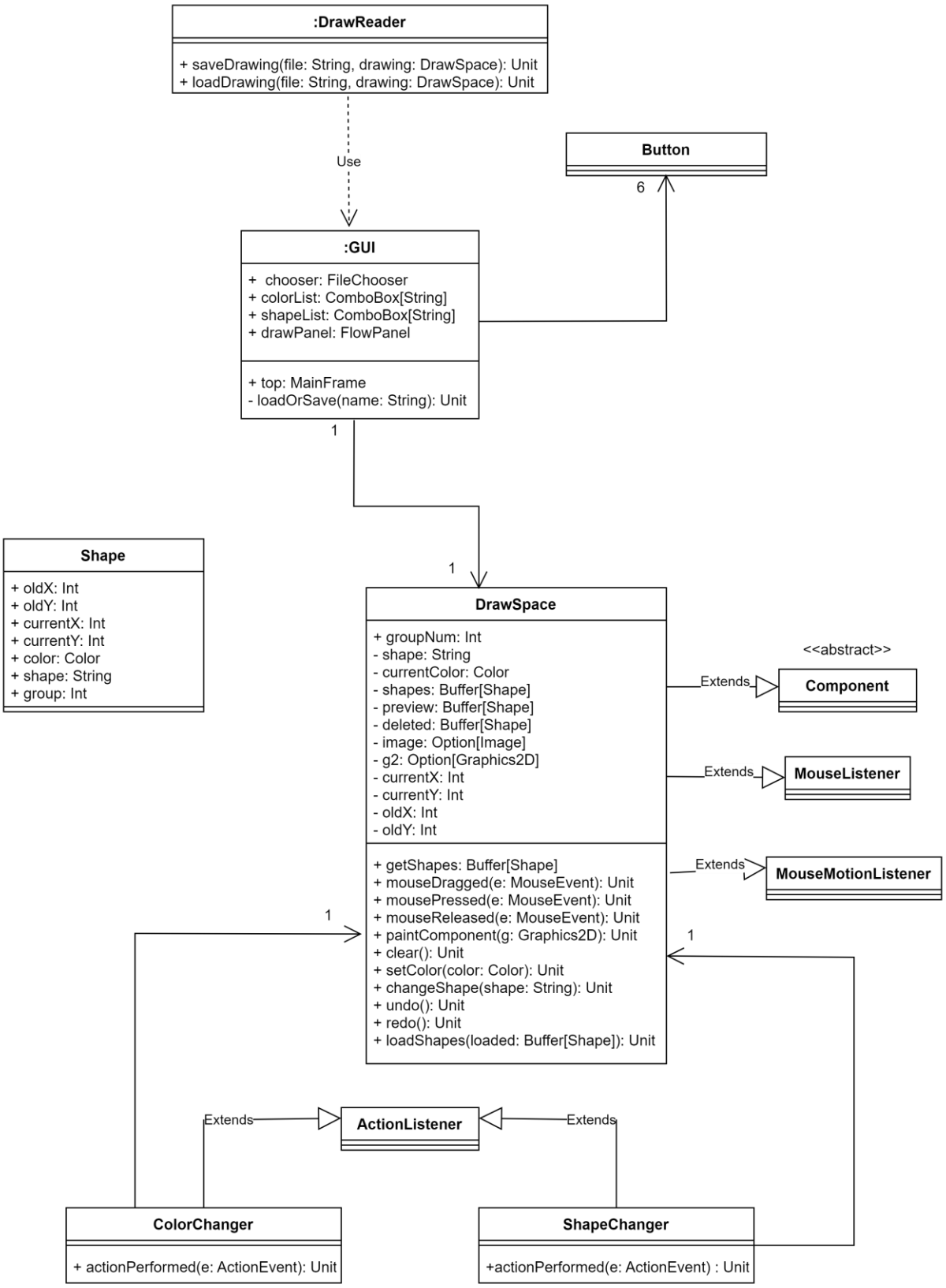
After you're done with your drawing you can save it to the desired file. Click the save button. A new window pops up. Here you see your own files from your computer. Type the file name where you want to save your drawing as you see down below and then click "Save". If the file does not exist, the program creates a new one. If the file exists, you can type its name and the program saves the data there. Another way to choose a file is to simply search for it and then click it.



Same goes for loading the drawing from existing file. Click the load button in the drawing program. After that a new load window pops up. Then you can choose the file. Keep in mind that the file needs to be a text file in order to load the data.

Program structure

UML


:DrawReader
+ saveDrawing(file: String, drawing: DrawSpace): Unit
+ loadDrawing(file: String, drawing: DrawSpace): Unit
Use
Button
6
:GUI
+ chooser: FileChooser
+ colorList: ComboBox[String]
+ shapeList: ComboBox[String]
+ drawPanel: FlowPanel
+ top: MainFrame
- loadOrSave(name: String): Unit
1
1
Shape
+ oldX: Int
+ oldY: Int
+ currentX: Int
+ currentY: Int
+ color: Color
+ shape: String
+ group: Int
DrawSpace
+ groupNum: Int
- shape: String
- currentColor: Color
- shapes: Buffer[Shape]
- preview: Buffer[Shape]
- deleted: Buffer[Shape]
- image: Option[Image]
- g2: Option[Graphics2D]
- currentX: Int
- currentY: Int
- oldX: Int
- oldY: Int
+ getShapes: Buffer[Shape]
+ mouseDragged(e: MouseEvent): Unit
+ mousePressed(e: MouseEvent): Unit
+ mouseReleased(e: MouseEvent): Unit
+ paintComponent(g: Graphics2D): Unit
+ clear(): Unit
+ setColor(color: Color): Unit
+ changeShape(shape: String): Unit
+ undo(): Unit
+ redo(): Unit
+ loadShapes(loaded: Buffer[Shape]): Unit
<<abstract>>
Extends
Component
Extends
MouseListener
Extends
MouseMotionListener
1
1
Extends
ActionListener
Extends
ColorChanger
+ actionPerformed(e: ActionEvent): Unit
ShapeChanger
+actionPerformed(e: ActionEvent) : Unit

This UML diagram showcases the most important parts of the final class structure of this project. Some classes and class parameters have been left out for the sake of readability. For example, classes ColorChanger and ShapeChanger use as parameters DrawSpace object and a ComboBox object. I also did not include custom exception class, that I used in DrawReader object, in this UML diagram.

The final program has all the necessary classes for dealing with the subproblems.

SWING

I think using Scala Swing was a good choice for this project. It offered some good classes to build my GUI. For example, Button class, ComboBox class and different panel classes. I learned how to create my own components that the GUI uses. For example, DrawSpace class which extends to component, MouseListener and MouseMotionListener. My GUI uses a FileChooser class for creating a window for file choosing. It was a good choice because it looks clean and enables users to choose the desired file where they want to save their drawing or load from.

Custom actionListeners

One of the essential things was the actionListeners that I had to add into buttons in order to them to response to users' clicks. One problem occurred when I wanted to add actionsListeners to GUI's ComboBoxes. Problem was that comboboxes had too many options and including them in method actionPerformed would make GUI's code 50 rows longer and readability would decrease so instead of doing so I coded two separate classes which are custom actionListeners for changing the color and the shape. Creating the classes was also a problem because, at the beginning, I used only drawSpace object as a parameter for the classes. When implementing the method actionPerformed, e.getSource brought anonymous type which was useless for match case implementation because it did not fit any known cases in match. I solved the problem by adding a new parameter (ComboBox) to the classes. Now I could use "combo.peer.getItemAt(combo.peer.getSelectedIndex)" which brough the desired outcome.

DrawSpace

I coded a separate class DrawSpace which is a custom component. This class handles the drawing, shows the changes to the user and creates shapes that are needed when saving or loading.

At first, I only extended the class to Component and added separate MouseAdapter and MouseMotionAdapter to the class. I had to override the mousePressed and mouseDragged methods like in the example down below (old solution). This was a big problem because these methods will be larger in later implementations. So, I had to change the structure of DrawSpace class by extending it to mouseListener and mouseMotionListener. This helped me to write a bit better mouseDragged and mouseReleased methods. With the new structure it was easier to separate implementation for pen method from line, circle and rectangle methods.

I also realized that it was not sensible to draw in these methods because I ran into a problem with keeping the information about the drawing. In other words, the old solution did not store any information about the drawing. So, I figured out that I need a class “shape” that stores the required information, which is later used in drawLine, drawOval and drawRectangle methods. This solution also helped me to store the information in a text-based file and coding the implementation for undo- and redo-methods. After this change instead of drawing the shapes in method mouseDragged the method paintComponent handled it.

Old solution

```
  peer.addMouseListener(new MouseAdapter {
  override def mousePressed(e: event.MouseEvent) = {
    oldX = e.getX
    oldY = e.getY
    }
  })

  peer.addMouseMotionListener(new MouseMotionAdapter {
    override def mouseDragged(e: event.MouseEvent) = {
      currentX = e.getX
      currentY = e.getY

      if (g2.isDefined) {
        g2.get.drawLine(oldX, oldY, currentX, currentY)
        repaint()
        oldX = currentX
        oldY = currentY
      }
    }
  })
}
```

Newer solution

```
      override def mousePressed(e: event.MouseEvent) = {
        oldX = e.getX
        oldY = e.getY
        }

       override def mouseDragged(e: event.MouseEvent) = {
         currentX = e.getX
         currentY = e.getY
         isDragged = true
         if (shape == "Pen") {
           //g2.get.drawLine(oldX, oldY, currentX, currentY)
           shapes += new Shape(oldX, oldY, currentX, currentY,
 currentColor, "Line")
           repaint()
           oldX = currentX
           oldY = currentY
```

After this I realized that my code was too repetitive. The code also had too many if-statements. So, I decided to change the structure a bit and instead of if-statements I used match case structure to get rid of unnecessary code. This solution made the code look better and I managed to shorten it by 10 to 15 rows. You can see from the last picture that I used methods min and abs when creating shapes. This solution helped me to reduce the code's length from approximately 40 lines down to 30 lines. Without these methods creating this match structure would have been too complicated.

Final solution

```
override def mouseDragged(e: event.MouseEvent) = {
  currentX = e.getX
  currentY = e.getY
  isDragged = true
  shape match {
    case "Pen"       => {
                        shapes += new Shape(oldX, oldY, currentX, currentY, currentColor, "Line", groupNum)
                        oldX = currentX
                        oldY = currentY
    }
    case "Line"      => preview += new Shape(oldX, oldY, currentX, currentY, currentColor, "Line", 0)
    case "Circle"    => preview += new Shape(min(oldX, currentX), min(oldY, currentY), abs(currentX - oldX), a
    case "Rectangle" => preview += new Shape(min(oldX, currentX), min(oldY, currentY), abs(currentX - oldX), a
    case "Ellipse"   => preview += new Shape(min(oldX, currentX), min(oldY, currentY), abs(currentX - oldX), a
    case _           => // does nothing
  }
  repaint()
```

DrawReader

The last thing that I implemented in my program was DrawReader object.

The object has two methods loadDrawing and saveDrawing. The methods need two parameters: file's name and the DrawSpace object. I decided to make DrawSpace one of the parameters for the method loadDrawing because the method needs to get the information somewhere and for the method saveDrawing because the method needs to access DrawSpace object in order to change the object's shape container with the loaded one.

Algorithms and data structures

When I was planning my project, I made some initial decisions on which data structures and algorithms I would use. Mutable buffer was the data structure choice, and I'd say it worked well for the project. I think that other containers, for example Stack, would work as well as buffer but I felt no need to use it because buffer provided all the methods and functionality that I needed for storing data.

On the algorithm side of things, there wasn't many algorithms that I needed in my project other than calculating the distance from oldX to currentX or from oldY to currentY in order to get the shape's desired width and length (currentX – oldX). I used a lot of ready-made methods that swing and java AWT provided.

Files and Internet access

My program doesn't use any external files from the Internet other than the saved files that are stored in the user's own computer.

My program uses text files for loading a drawing. After some thinking I decided to use nearly the same format as in the OS2 assignment HumanWritableIO. The format is easy to read and write as well. I came up with this:

```
Shapes:
Type: Line
x-coordinate: 311
y-coordinate: 150
Width: 204
Height: 359
Color: 0,0,0
Group: 0
------
Type: Circle
x-coordinate: 147
y-coordinate: 426
Width: 172
Height: 172
Color: 0,255,0
Group: 0
------
EndOfFile
```

Make sure that there is "EndOfFile" at the end and "Shapes:" at the start of the file because otherwise the program won't load the data. Keep in mind the parameters must be written the same way as in the example above. The program doesn't need any setup files in order to start to work.

Note: While trying to load a file that is not in the correct format. The program does nothing. The only thing that happens is that console will print exception's message. If you don't give the right number of parameters, it will print this message:

```
Not enough parameters for creating a shape: Instead of 7 parameters you gave only 6
```

Testing

My original testing plan involved only testing all the methods with the help of my GUI. I tested all my methods with GUI because with its help it was easy to determine if there was a problem in the method.

Everything was visual. For example, when I tested my paintComponent method and tried to draw an oval shape, at first, it did not draw it as I planned. So, by fixing the method and then booting the GUI I managed to test all my methods and fix them until they worked.

Only in DrawReader object when testing loadDrawing method I used printing as well as my GUI. Printing was efficient in checking what was in the data buffer after the DrawReader has read contents of the file.

Known bugs and missing features

I'm not sure if there are any bugs left that I'm aware of in my program.

3 Best sides and 3 weaknesses

I think that I implemented drawSpace and my GUI quite well, so I think they are the best sides of the program. Other parts of the program are also good. I'd say that one weakness comes to my mind, and it is my drawReader object because I had little to no time to code it, so I didn't have much time to make the code look a little bit better there, I guess.

Deviations from the plan, realized process and schedule

First, I made the GUI so that I can test next methods. Creating the GUI took me longer than I thought. I planned it to take me only few days but in reality it took more than 10 days. After GUI, I created DrawSpace and it took me the longest, over two weeks because it had so many methods I needed to code. After DrawSpace, I started to code ColorChanger and ShapeChanger. This took me about a week, I think. The last two things were DrawReader and a FileChooser. Implementing them took me about one day because I was in a hurry.

I learned a lot during this project. When the project started, I had no idea how swing, awt or filechooser works. So, I think, this project was a pleasant experience in terms of learning new stuff.

The process of building the program did not differ from the plan but the time that I had to use was different. Also, other school projects took so much time from me.

Final evaluation

All In all, I'm satisfied with my project. Everything I coded works well so far. Good aspects of the program are the GUI and DrawSpace. The code in them is clean and works well. One of the shortcomings is that the starting point of the circles is not its center point. I think it could be fixed by adding some sort of a calculation method which calculates the center point of the circle.

On the data structure side of things, I could use Stack instead of Buffer but I think it would not bring any added value to the program. The structure would've been the same.

If I started the project again from the beginning, I would have paid more attention to the appearance of the GUI and would've made it look better.

References

https://www.javatpoint.com/java-jbutton

https://www.scala-lang.org/api/2.12.5/scala-swing/scala/swing/

https://www.javatpoint.com/java-jcomponent#:~:text=The%20JComponent%20class%20is%20the,JScrollPane%2C%20JPanel%2C%20JTable%20etc.&text=The%20JComponent%20class%20extends%20the%20Container%20class%20which%20itself%20extends%20Component.

https://docs.oracle.com/javase/7/docs/api/javax/swing/JButton.html

https://docs.oracle.com/javase/tutorial/uiswing/events/actionlistener.html

https://docs.oracle.com/javase/tutorial/uiswing/components/combobox.html

https://www.tutorialspoint.com/swing/swing_action_listener.htm#:~:text=The%20class%20which%20processes%20the,using%20the%20addActionListener()%20method.

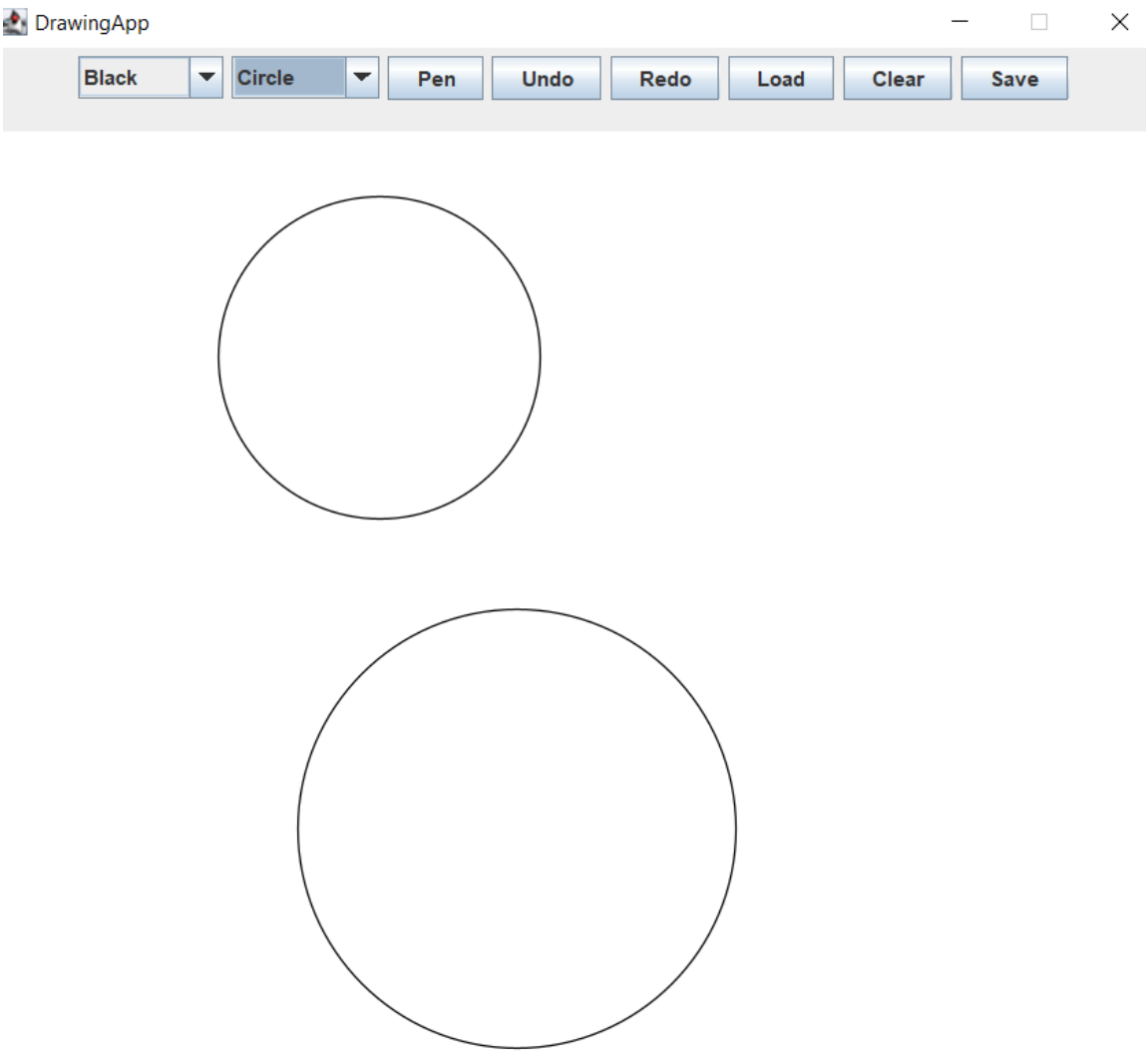
https://www.tabnine.com/code/java/methods/java.awt.Graphics/drawRect

http://thushw.blogspot.com/2015/09/scala-collectfirst-example.html

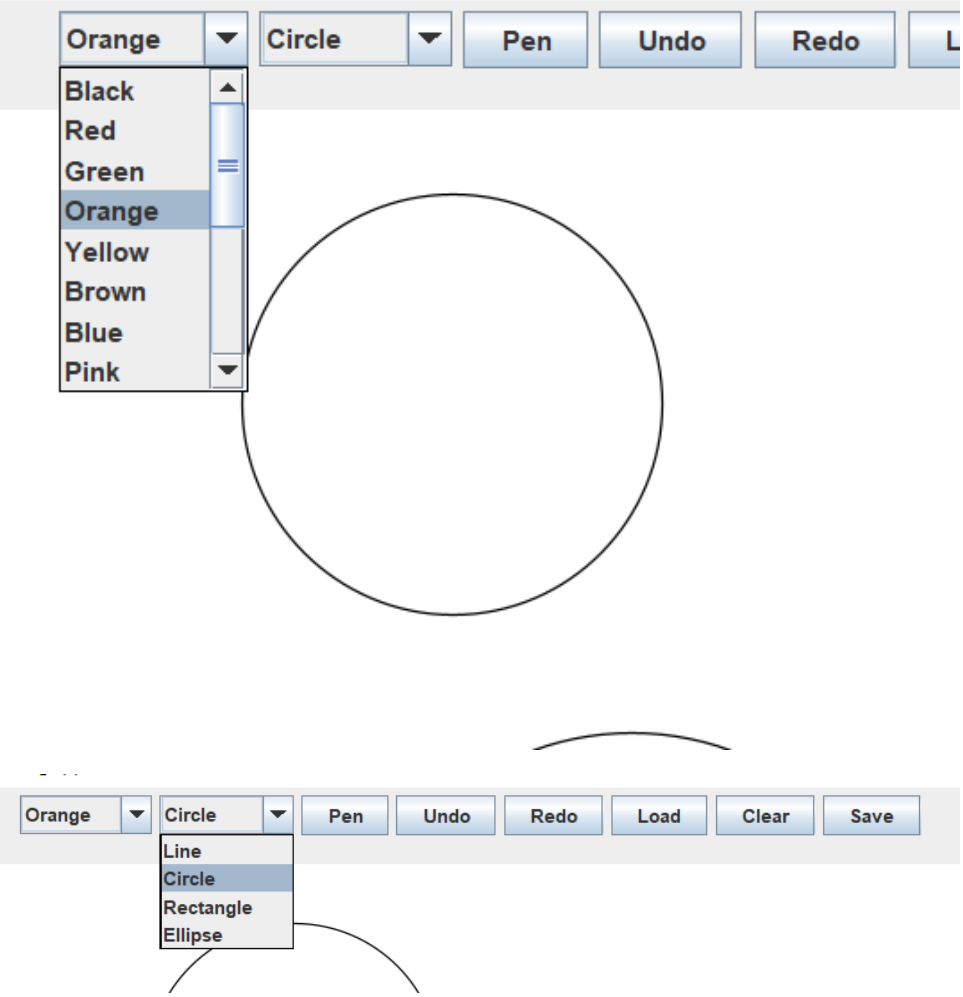
https://examples.javacodegeeks.com/desktop-java/swing/jfilechooser/create-file-chooser-dialog/

https://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html

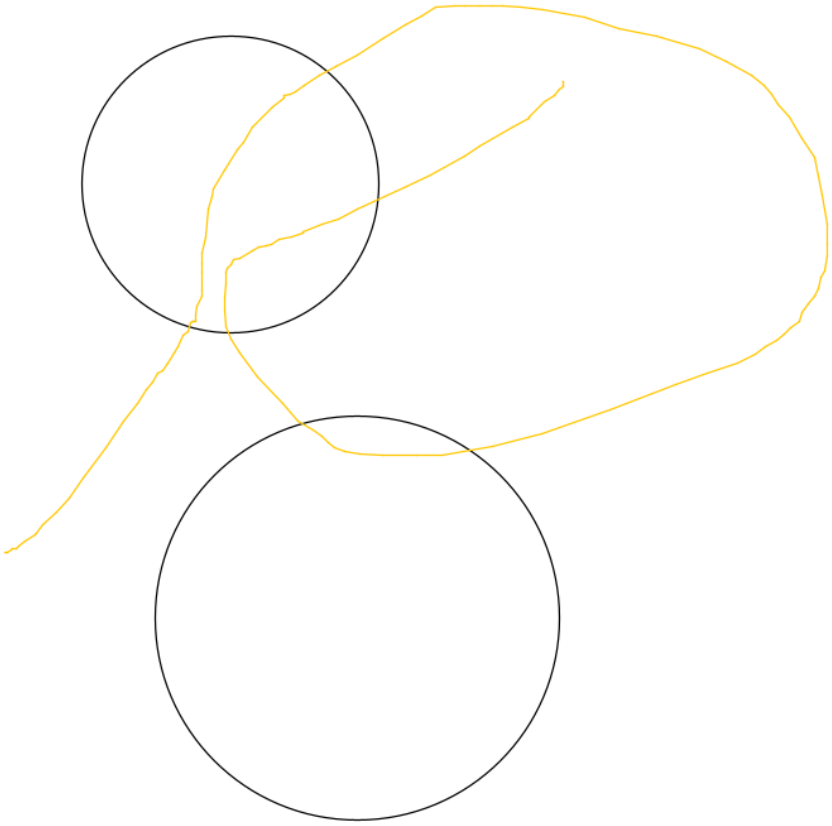
Appendixes



Draw different shapes by clicking your mouse and dragging it.



Change the color and the shape by clicking these catalogs and click the desired option.



By clicking the pen button, you can draw freely.