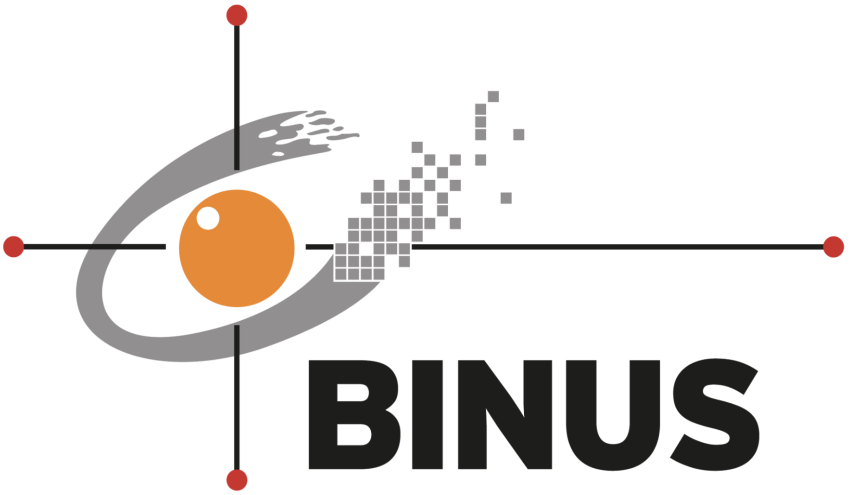
OBJECT-ORIENTED PROGRAMMING

Final Project Report

Lecturer: Jude Joseph Lamug Martinez D4017

Course Name: Object-Oriented Programming

Course Code: COMP 6699001



BINUS UNIVERSITY

Kimberly Mazel

Class: L2BC

2502022250

Faculty of Computing and Media

Computer Science

Binus International University

Even Semester 2022

Table of Contents

1. Project Specification

1.1. Objective

People are more well-versed in art than they think. Many people have seen a wide range of artworks yet do not know anything about it. If they were to be asked if they knew a certain painting by its name, they would say no. However, once they actually see the painting, they realize that they have seen it before, but they just didn't know what it was called or who the artist is. This project aims to allow users to scan images of paintings and display the matched painting along with its information, through image processing. Users can also find artwork information based on their name, or add non-existing artworks to the database. This project is specifically directed to Vincent Van Gogh and his artworks, hence, its name Van Go! Both paintings and sketches from his letters are included.

1.2. Concept

The user will be presented with two options when running the program:

a) Paintings
b) Letters

The respective JFrame window will be run when the user selects an option.

Both options allow the same functions:

- Scanning and finding an artwork by inputting an image
- Adding a new artwork to the program
- Removing an artwork from the program
- Finding an artwork by its name

On each JFrame there are labeled buttons that allow the user to make use of the mentioned functions.

The options differ in their attributes and respective methods. As this is an object-oriented project, there is a parent file for artworks and two child files, one for paintings and another for letters.

2. Solution Design

2.1. Class Diagram

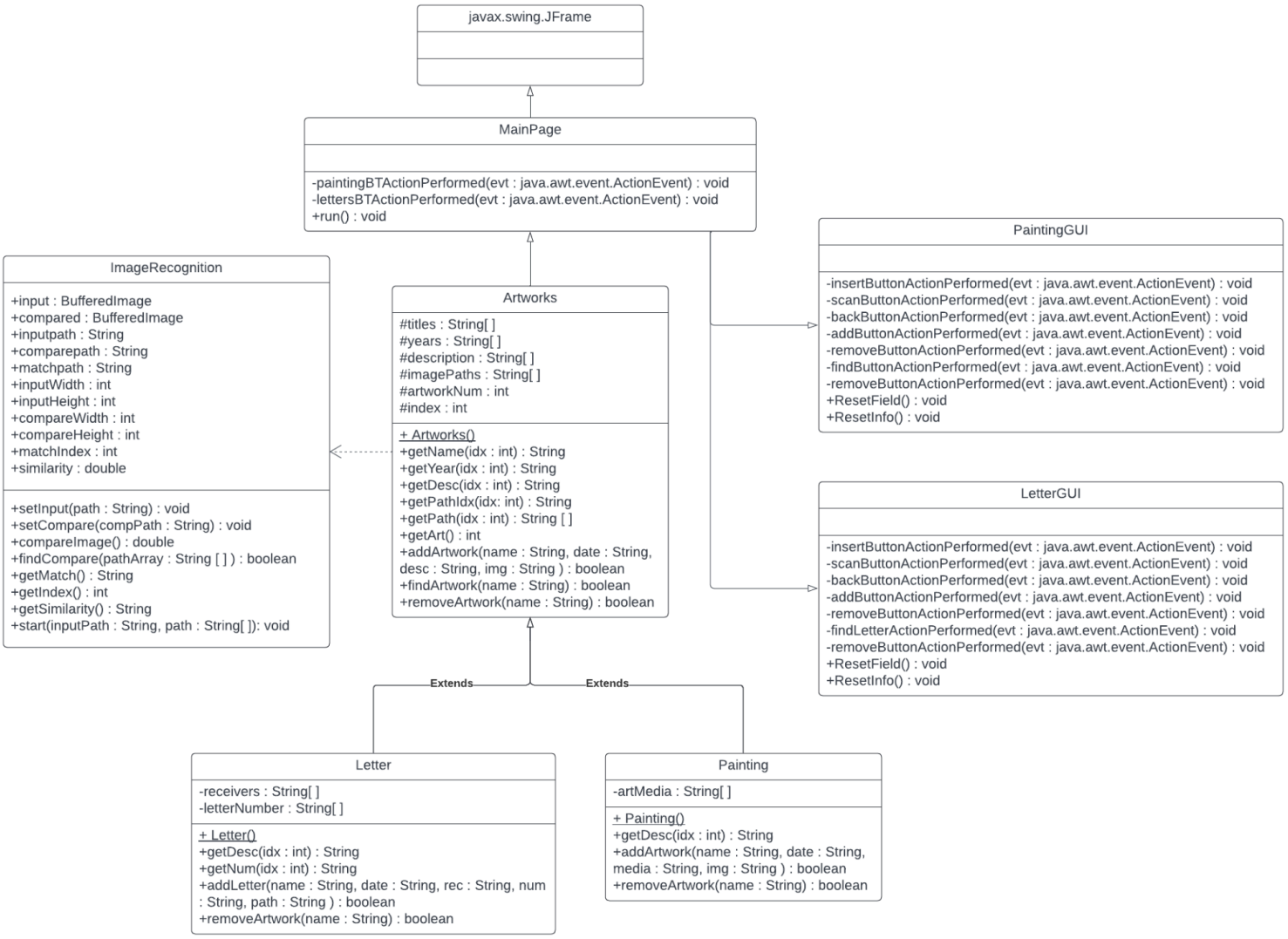


Image 2.1 Class diagram of the program

2.2. Code Implementation

a) Artworks.java

This is the parent file. As shown in the class diagram, it is the parent of Letter.java and Painting.java. It has 4 ArrayList variables and 2 integer variables. All the variables are protected as they are also used in the children files, meaning that they are accessible by the other files in the package but not anywhere else.

Artworks.java's attributes:

- titles (ArrayList for strings of artwork names)
- years (ArrayList for strings of artwork years)
- description (ArrayList for strings of artwork descriptions, for example, the type of artwork)
- imagePaths (ArrayList for strings of artwork image paths)
- artworkNum (Integer value of the number of artworks in the program)
- index (Integer value for the index of the ArrayList, used in the methods)

```
// ARRAYLIST FOR INFORMATIONS
protected ArrayList<String> titles; // TITLES
protected ArrayList<String> years; // YEARS
protected ArrayList<String> description; // DESCRIPTIONS
protected ArrayList<String> imagePaths; // IMAGE PATHS

protected int artworkNum = 0; // NUMBER OF ARTWORKS
protected int index = 0; // INDEX OF INFORMATION
```

Image 2.2 Attributes of Artworks.java

Artwork.java's methods:

i. Artworks()

- Constructor that creates the 4 ArrayLists and adds "N/A" to each of them. This is for artworks that are not available in the ArrayList. For example, if an artwork that is scanned is not available, the strings in the ArrayLists at index 0 will be used.

```
public Artworks(){
    // ARRAYLIST FOR TITLES
    titles = new ArrayList<String>();
    titles.add("Artwork N/A"); // FOR ARTWORKS NOT AVAILABLE

    years = new ArrayList<String>();
    years.add("N/A"); // FOR ARTWORKS NOT AVAILABLE

    description = new ArrayList<String>();
    description.add("N/A"); // FOR ARTWORKS NOT AVAILABLE

    imagePaths = new ArrayList<String>();
    imagePaths.add(""); // FOR ARTWORKS NOT AVAILABLE

    index++; // AT INDEX 0 IS INFORMATION FOR IF ARTWORK IS NOT AVAILABLE
}
```

Image 2.3 Artworks.java constructor

ii. Getters

- getName(int idx), getYear(int idx), getDesc(int idx), getPathIdx(int idx), getPath(), getArt()
- These are all getter methods to access values from the code.
- For the methods that have idx as a parameter, it returns the value found at that index of each ArrayList. For example, getName(1) returns the value at index 1 of the titles ArrayList.
- getPathIdx() gets the path at the inputted index, while getPath returns the imagePaths() ArrayList as a whole.
- getArt() returns the value of the index variable. In this program, the index variable is used to identify which artworks are to be removed.

```
// GET NAME OF IMAGE
public String getName(int idx){
    return titles.get(idx);
}

// GET YEAR OF IMAGE
public String getYear(int idx){
    return years.get(idx);
}

// GET DESCRIPTION OF IMAGE
public String getDesc(int idx){
    return description.get(idx);
}
```

```
// GET ONE PATH
public String getPathIdx(int idx){
    return imagePaths.get(idx);
}

// GET ALL PATHS
public ArrayList<String> getPath(){
    return imagePaths;
}

// GET INDEX
public int getArt(){
    return index;
}
```

Image 2.4 Getter methods of Artworks.java

iii. addArtwork(String name, String year, String desc, String img)

- This method is to add artworks to the ArrayLists. It takes in the name, year, description, and image path of the artwork to add to the respective ArrayLists. The name is always converted to upper case before appending, to make the presence checks case-insensitive.
- There is an if-else statement that prevents duplicates of artworks. If the artwork already exists in the ArrayLists, the method will return false.
- If the artwork does not exist in the ArrayLists, each input from the parameters will be added to the ArrayLists. The artworkNum variable is incremented so the total of artworks is updated, and the method returns true.

```
public boolean addArtwork(String name, String date, String desc,
        String img){
    // IF ARTWORK ALREADY EXISTS, RETURN FALSE
    // toUpperCase() used so that it is case insensitive
    if(titles.contains(name.toUpperCase()) == true){
        return false;
    }

    // IF ARTWORK DOES NOT EXIST, RETURN TRUE AND ADD TO ARRAY
    else{
        // toUpperCase() used so that it is case insensitive
        titles.add(name.toUpperCase()); // ADDS TITLE
        years.add(date); // ADDS YEAR
        description.add(desc); // ADDS DESCRIPTION
        imagePaths.add(img); // ADDS IMAGE PATH

        artworkNum++; // NUMBER OF ARTWORKS + 1

        return true;
    }
}
```

Image 2.4 addArtwork method

iv. findArtwork(String name)

- This method is used to find artworks in the ArrayList. It takes in the name of the artwork and returns true or false depending on if it exists or not.
- Similar to addArtwork, it checks if it exists in the array using an if-else statement, and returns false if it does not.

- If it does exist, the index variable is updated with the index of the inputted name in the titles ArrayList.
- This index is used in the JForm file to update the labels with the corresponding information.

```
public boolean findArtwork(String name){
    // IF ARTWORK IS NOT AVAILABLE IN DATABASE RETURN FALSE
    // toUpperCase() used so that it is case insensitive
    if(titles.contains(name.toUpperCase()) == false){
        return false;
    }

    // IF ARTWORK IS AVAILABLE, RETURN TRUE AND INFORMATION
    else{
        // GETS INFORMATION BASED ON INDEX
        // toUpperCase() used so that it is case insensitive
        index = titles.indexOf(name.toUpperCase());
        return true;
    }
}
```

Image 2.5 findArtwork method

v. removeArtwork(String name)

- This method is used to remove artworks from the ArrayList. It works similarly to findArtwork but removes the values instead of only updating the index.
- There is an if-else statement as well to ensure that the artwork exists in the array.
- The method returns true after the values are removed from each ArrayList by the index.
- The artworkNum variable is decremented to update the total number of artworks.

```
public boolean removeArtwork(String name){
    // IF ARTWORK IS NOT AVAILABLE RETURN FALSE
    // toUpperCase() used so that it is case insensitive
    if(titles.contains(name.toUpperCase()) == false){
        return false;
    }

    // IF ARTWORK IS AVAILABLE, RETURN TRUE AND REMOVE
    else{
        // FIND INFORMATION BASED ON INDEX
        // toUpperCase() used so that it is case insensitive
        index = titles.indexOf(name.toUpperCase());

        // REMOVE INFORMATION FROM ARRAYLISTS
        // toUpperCase() used so that it is case insensitive
        titles.remove(name.toUpperCase());
        years.remove(years.get(index));
        description.remove(description.get(index));
        imagePaths.remove(imagePaths.get(index));

        // ARTWORK NUMBER - 1
        artworkNum--;

        return true;
    }
}
```

Image 2.6 removeArtwork method

b) Letter.java

One of the children files of Artwork.java. It inherits the attributes and has two additional ArrayLists, one for the receiver of the letter and another for the letter number.

```
public class Letter extends Artworks {

    private ArrayList<String> receivers; // LETTER RECEIVERS
    private ArrayList<String> letterNumber; // LETTER NUMBERS
```

Image 2.7 Letter.java's attributes

Letter.java's methods:

i. Getters

- The getDesc method from the parent class is overridden so that the receivers ArrayList is used instead of the description ArrayList.

- The getNum method is used to retrieve the letter number based on the inputted index from it's parameter.

```
@Override
// GET RECEIVER OF IMAGE
public String getDesc(int idx){
    return receivers.get(idx);
}
```

```
// GET NUMBER OF LETTER
public String getNum(int idx){
    return letterNumber.get(idx);
}
```

Images 2.8 Getter methods of Letter.java

ii. addLetter(String name, String date, String rec, String num, String path)

- This function is similar to addArtwork except it takes in more parameters as instances of Letter.java have receivers and letter numbers as well.
- It has the if-else statement to ensure that there are no duplicates and returns true once it is done adding the values to the respective ArrayLists.

```
public boolean addLetter(String name, String date,
        String rec, String num, String path){
    // IF ARTWORK ALREADY EXISTS, RETURN FALSE
    // toUpperCase() used so that it is case insensitive
    if(titles.contains(name.toUpperCase()) == true){
        return false;
    }

    // IF ARTWORK DOES NOT EXIST, RETURN TRUE AND ADD TO ARRAY
    else{
        // toUpperCase() used so that it is case insensitive
        titles.add(name.toUpperCase()); // ADDS TITLE
        years.add(date); // ADDS YEAR
        receivers.add(rec); // ADDS RECEIVER
        letterNumber.add(num); // ADDS LETTER NUMBER
        imagePaths.add(path); // ADDS IMAGE PATH

        artworkNum++; // NUMBER OF ARTWORKS + 1

        return true;
    }
}
```

Image 2.9 addLetter method

iii. Overridden removeArwork(String name)

- This method is overridden from the parent file. There are additional ArrayLists that the values are removed from, which are the receivers and letterNumber variables.

```
@Override
public boolean removeArtwork(String name){
    // IF ARTWORK IS NOT AVAILABLE RETURN FALSE
    // toUpperCase() used so that it is case insensitive
    if(titles.contains(name.toUpperCase()) == false){
        return false;
    }

    // IF ARTWORK IS AVAILABLE, RETURN TRUE AND REMOVE
    else{
        // FIND INFORMATION BASED ON INDEX
        // toUpperCase() used so that it is case insensitive
        index = titles.indexOf(name.toUpperCase());

        // REMOVE INFORMATION FROM ARRAYLISTS
        // toUpperCase() used so that it is case insensitive
        titles.remove(name.toUpperCase());
        years.remove(years.get(index));
        receivers.remove(receivers.get(index));
        letterNumber.remove(letterNumber.get(index));
        imagePaths.remove(imagePaths.get(index));

        // ARTWORK NUMBER - 1
        artworkNum--;

        return true;
    }
}
```

Image 2.10 removeArtwork overridden method

c) Painting.java

The other child of the Artwork class. It also inherits the attributes but has an additional ArrayList variable for the media of the artwork, for example, oil paint or watercolor.

```
public class Painting extends Artworks {

    private ArrayList<String> artMedia; // ART MEDIA
```

Image 2.11 Painting.java's attributes

Painting.java's methods:

i. Overridden getDesc method

- The getDesc method from the parent class is overridden so that the artMedia ArrayList is used instead of the description ArrayList

```
@Override
// GET MEDIA OF IMAGE
public String getDesc(int idx){
    return artMedia.get(idx);
}
```

Image 2.12 Overriden getDesc method of Painting.java

ii. Overridden addArtwork and removeArtwork

- Both methods from the parent class are overridden so that artMedia variable is used instead of the description variable.
- The function of the methods is the same as the parent class, there are only changes in the variables used.

```
@Override
public boolean addArtwork(String name, String date, String media,
        String img){
    // IF ARTWORK ALREADY EXISTS, RETURN FALSE
    // toUpperCase() used so that it is case insensitive
    if(titles.contains(name.toUpperCase()) == true){
        return false;
    }

    // IF ARTWORK DOES NOT EXIST, RETURN TRUE AND ADD TO ARRAY
    else{
        // toUpperCase() used so that it is case insensitive
        titles.add(name.toUpperCase()); // ADDS TITLE
        years.add(date); // ADDS YEAR
        artMedia.add(media); // ADDS MEDIA
        imagePaths.add(img); // ADDS IMAGE PATH

        artworkNum++; // NUMBER OF ARTWORKS + 1

        return true;
    }
}
```

Image 2.13 Overriden addArtwork method of Painting.java

```java
@Override
public boolean removeArtwork(String name){
    // IF ARTWORK IS NOT AVAILABLE RETURN FALSE
    // toUpperCase() used so that it is case insensitive
    if(titles.contains(name.toUpperCase()) == false){
        return false;
    }

    // IF ARTWORK IS AVAILABLE, RETURN TRUE AND REMOVE

    // FIND INFORMATION BASED ON INDEX
    // toUpperCase() used so that it is case insensitive
    index = titles.indexOf(name.toUpperCase());

    // REMOVE INFORMATION FROM ARRAYLISTS
    titles.remove(index);
    years.remove(index);
    artMedia.remove(index);
    imagePaths.remove(index);

    // ARTWORK NUMBER - 1
    artworkNum--;

    return true;
}
```

Image 2.14 Overriden removeArtwork method of Painting.java

2.3. JForm Interface

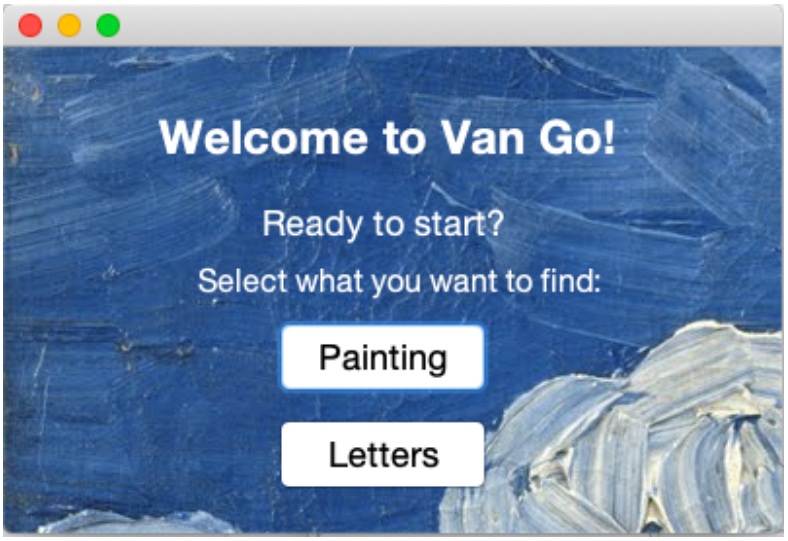
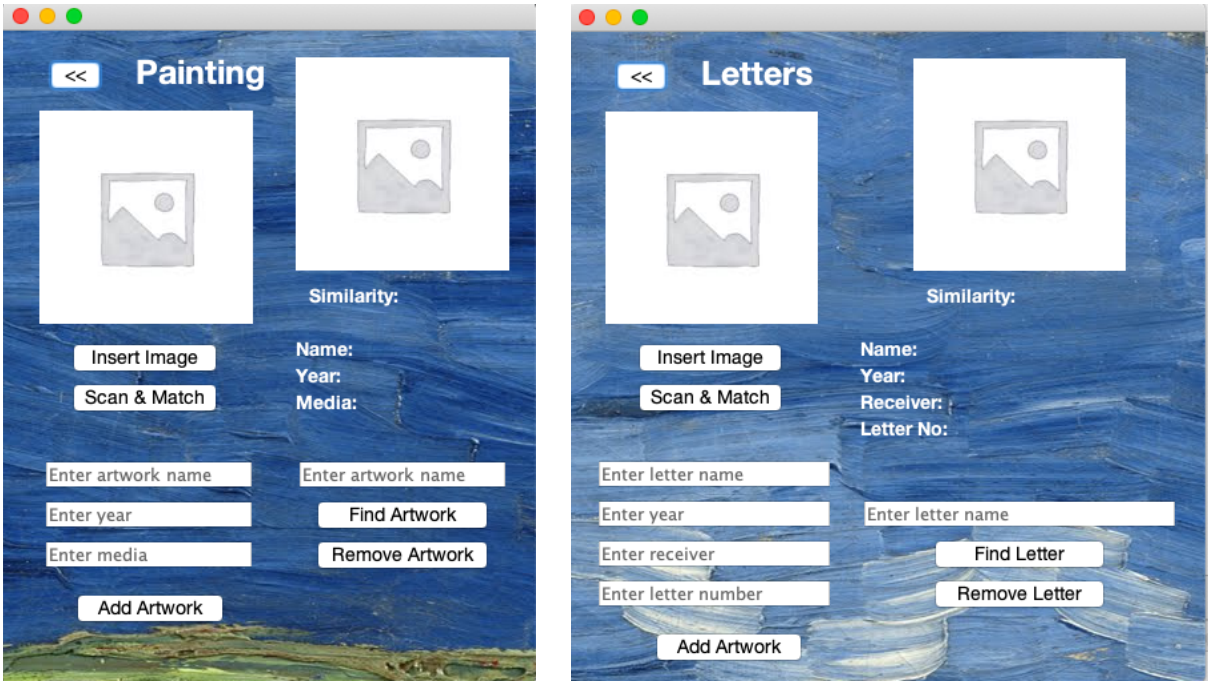


Image 2.15 Opening GUI page



Images 2.16 Painting and Letters GUI

The GUIs are made out of JFrames that have buttons, fields, and labels that allow the user to see the methods of the classes at work. In the painting and letters GUI, the two image placeholders are the places where the inputted artwork image (on the left) and the matched artwork image (on the right) will be.

Functions of GUI:

a) Scan & Match
   - The user inputs an image by clicking on “Insert Image” button
   - When “Scan & Match” button is clicked, the matched image will appear on the right image placeholder, along with the similarity percentage, name, year, and media of the matched painting, or name, year, receiver, and letter number for the matched letter.

b) Add Artwork
   - The user can input an image using the “Insert Image” button
   - The fields below (“Enter artwork name”, “Enter year”, “Enter media” or “Enter receiver” and “Enter letter number”) can be filled.
   - When the user presses the “Add Artwork” button, the addArtwork (or the overridden methods) is called.

c) Find Artwork
   - Users can find artworks and their information using the artwork's name.
   - The name should fill the field above the "Find Artwork" button and once the button is pressed, the findArtwork method is called.

d) Remove Artwork
   - Users can remove an artwork from the program by filling in the field above the button.
   - The "Remove Artwork" button will then call the removeArtwork function or its respective overridden methods.

2.4. Error Catching

- Each button and its respective function has an if statement to prevent duplicates or to check if the artwork exists or not. The user cannot remove an artwork that does not exist or add an artwork that is already there. There will be a message at the bottom left of the interface to inform them.

```
// INFORMS USER ON ACTIONS
// IF ARTWORK NOT FOUND
if (painting.removeArtwork(name)==false){
    warningLabel.setText("Artwork not found.");
}

// IF ARTWORK FOUND AND REMOVED
else{
    painting.removeArtwork(name);
    warningLabel.setText("Artwork removed.");
}
```

Image 2.17 If else statements for messages

- The "Input Image" button has been programmed to only allow images. The user cannot input any other format. This is done by importing javax.swing.filechooser and using FileNameExtensionFilter. The filter is then applied to the button using setFileFilter.

```
// INSERT BUTTON ACTIONS (WHEN CLICKED)
private void insertButtonActionPerformed(java.awt.event.ActionEvent evt) {
    warningLabel.setText("");
    JFileChooser browseInput = new JFileChooser();

    // FILE FILTER (ONLY ALLOWS IMAGES)
    FileNameExtensionFilter filter = new FileNameExtensionFilter("IMAGES",
            "png", "jpg", "jpeg");
    browseInput.setFileFilter(filter);

    int showDialogue = browseInput.showOpenDialog(null);

    if (showDialogue == JFileChooser.APPROVE_OPTION) {
        File selectInput = browseInput.getSelectedFile();
        inputPath = selectInput.getAbsolutePath();

        // GETTING IMAGE FROM FILE PATH
        ImageIcon img = new ImageIcon(inputPath);

        // RESIZE TO VIEW'S SIZE
        int width = inputView.getWidth();
        int height = inputView.getHeight();
        Image resized = img.getImage().getScaledInstance(width, height,
                Image.SCALE_DEFAULT);

        // VIEW IMAGE IN INPUTVIEW
        inputView.setIcon(new ImageIcon(resized));
    }
}
```

Image 2.18 Insert Button code

In the if statements of the functions, the name is made to upper case using toUpperCase() so that finding or removing an artwork would be case-insensitive. This means that it doesn't matter whether or not the inputted name is in capitals or not. This is because the inputted names are always converted to upper case in the addArtwork function, therefore the checking methods should be in the same cases.

```
if(titles.contains(name.toUpperCase()) == false){
    return false;
}
```

Image 2.19 Snippet of if statement in the findArtwork method

In the snippet, the getter functions return the respective value based on the index, which is obtained from the getArt() function. There are also reset functions that are made to revert the fields, labels, and images to their default.

```
// RESET INPUT FIELDS
public void ResetField(){
    nameField.setText("Enter name");
    yearField.setText("Enter year");
    descField.setText("Enter media");
}

// RESET INFO
public void ResetInfo(){
    // RESETS INFORMATION LABELS
    artSimilarity.setText("Similarity: ");
    artName.setText("Name: ");
    artDesc.setText("Media: ");
    artYear.setText("Year: ");

    // RESET MATCHVIEW TO PLACEHOLDER
    ImageIcon find = new ImageIcon("src/artscanner/images/placeholder.jpeg");
    Image resize = find.getImage().getScaledInstance(matchView.getWidth(),
            matchView.getHeight(), Image.SCALE_DEFAULT);
    matchView.setIcon(new ImageIcon(resize));
}
```

Image 2.20 Reset functions

2.5. ImageRecognition.java

This code is responsible for comparing two images and returning the similarity. The code in my program is referenced from GeeksforGeeks. Besides the setters and getters, there are two major methods, which are compareImage and findCompare. The first method consists of the calculations resulting in the similarity value in the form of a double. The second method consists of a for loop that compares the input image with each image saved in the ArrayList of image paths. The compared image is set by using the image with a similarity of at least 85%.

```
public boolean findCompare(ArrayList<String> pathArray) throws IOException {
    for(int i=1; i<pathArray.size(); i++){
        setCompare(pathArray.get(i));
        compareImage();

        if(similarity>=85){
            matchpath = pathArray.get(i);
            matchIndex = i;
            return true;
        }
    }
    return false;
}
```

Image 2.21 findCompare method

```
public double compareImage(){
    // IF WIDTH AND HEIGHT IS DIFFERENT, SIMILARITY IS 0
    // CANNOT COMPARE PIXELS IF WIDTH AND HEIGHT IS DIFFERENT
    if((inputWidth != compareWidth) || (inputHeight != compareHeight)){
        similarity = 0;
    }

    // ELSE COMPARE THE IMAGES
    else{

        // DIFFERENCE OF PIXELS IN IMAGES
        long difference = 0;

        // FOR ALL PIXELS IN HEIGHT OF IMAGE
        for(int i=0; i < inputHeight; i++){

            // FOR ALL PIXELS IN WIDTH OF IMAGE
            for(int j=0; j < inputWidth; j++){

                // RGB OF INPUT AND COMPARED IMAGE
                int rgbInput = input.getRGB(j, i);
                int rgbCompared = compared.getRGB(j, i);

                // INPUTTED IMAGE
                int redInput = (rgbInput >> 16) & 0xff; // RED OF INPUT
                int greenInput = (rgbInput >> 8) & 0xff; // GREEN OF INPUT
                int blueInput = (rgbInput)&0xff; // BLUE OF INPUT

                // COMPARED IMAGE
                int redCompared = (rgbCompared >> 16) & 0xff; // RED OF COMPARED
                int greenCompared = (rgbCompared>> 8) & 0xff; // GREEN OF COMPARED
                int blueCompared = (rgbCompared)&0xff; // BLUE OF COMPARED

                // DIFFERENCES OF RED GREEN AND BLUE PIXELS
                difference += Math.abs(redInput - redCompared);
                difference += Math.abs(greenInput - greenCompared);
                difference += Math.abs(blueInput - blueCompared);
            }
        }
```

Image 2.22 Part of compareImage that calculates the differences of each pixel

```
        // CALCULATE THE TOTAL PIXELS AND DIFFERENCES
        double totalPixels = inputWidth * inputHeight * 3;
        double averageDifference = difference / totalPixels;

        // PERCENTAGE OF DIFFERENCE
        double percentage = (averageDifference / 255) * 100;

        // PERECENTAGE OF SIMILARITY TO 2 DECIMAL PLACE
        similarity = 100-percentage;
        }

        return similarity;
```

Image 2.23 Snippet of compareImage that calculates the similarity

The rest of the methods are standard getters and setters except for setInput and setCompare. This is because it takes in an image path and makes use of ImageIO.read() and the java.io.File to get the image from the path.

```
public void setInput(String path) throws IOException{
    inputpath = path;
    File inputimg = new File(inputpath);
    input = ImageIO.read(inputimg);
    inputWidth = input.getWidth();
    inputHeight = input.getHeight();
}

public void setCompare(String compPath) throws IOException{
    comparepath = compPath;
    File comparedimg = new File(comparepath);
    compared = ImageIO.read(comparedimg);
    compareWidth = compared.getWidth();
    compareHeight = compared.getHeight();
}
```

Image 2.23 setInput and setCompare methods

3. Evidence of Working Program

a) Scan & Match Evidence

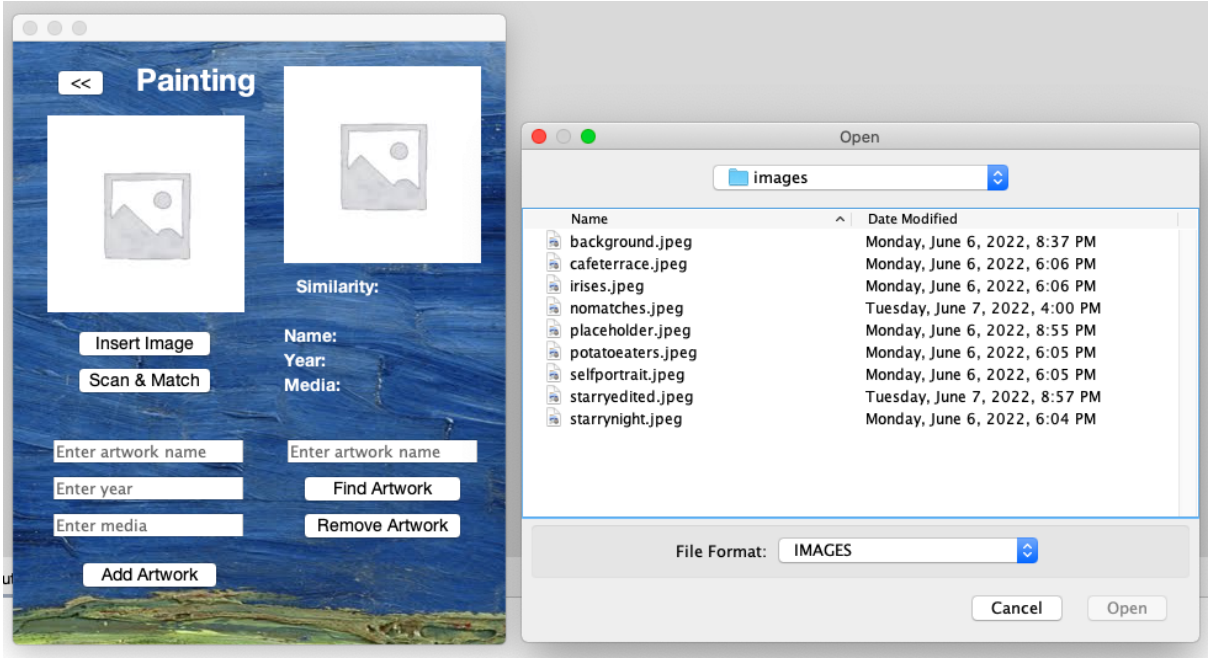
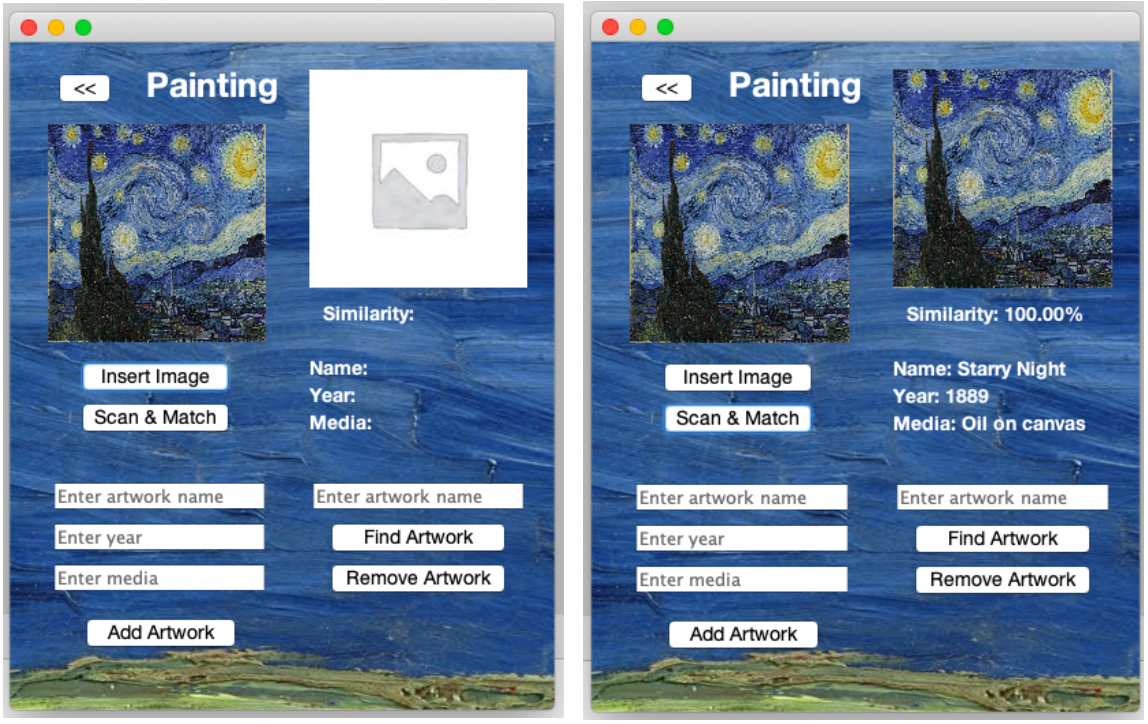


Image 3.1 Dialog box after pressing the “Insert Image” button

As seen in the image above, there is a “starrynight.jpeg” file that will be demonstrated in the next images below. (All images are from Wikipedia).



Images 3.2 On the left is the result of adding “starrynight.jpeg”, on the right is the result of clicking the “Scan & Match” button

The match view’s image box changes, and the similarity, name, year and media labels are filled accordingly. The inputted image is the same with the one in the ArrayList, which results in the 100% similarity. The next images will be with a slightly edited version of the image, which is “starryedited.jpeg” in image 3.1.

Image 3.3 Result of “starryedited.jpeg”

As shown in the image, the similarity has changed. As the program only allows matches with a similarity of at least 85%, the matches are as accurate as possible while still staying lenient to small changes, for example, in saturation or brightness.

b) Images Only Input Evidence

As stated before, the project is programmed to only take in images. The following images will show that the user is not allowed to choose any other non-image formats

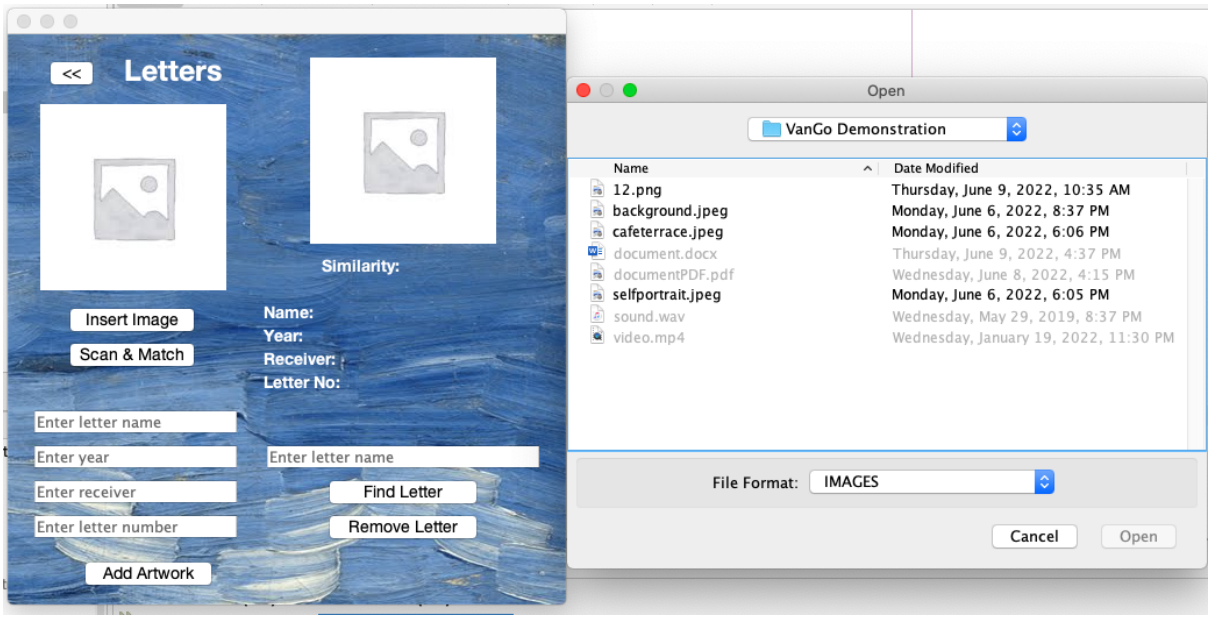


Image 3.4 The non-image formats are unable to be picked.

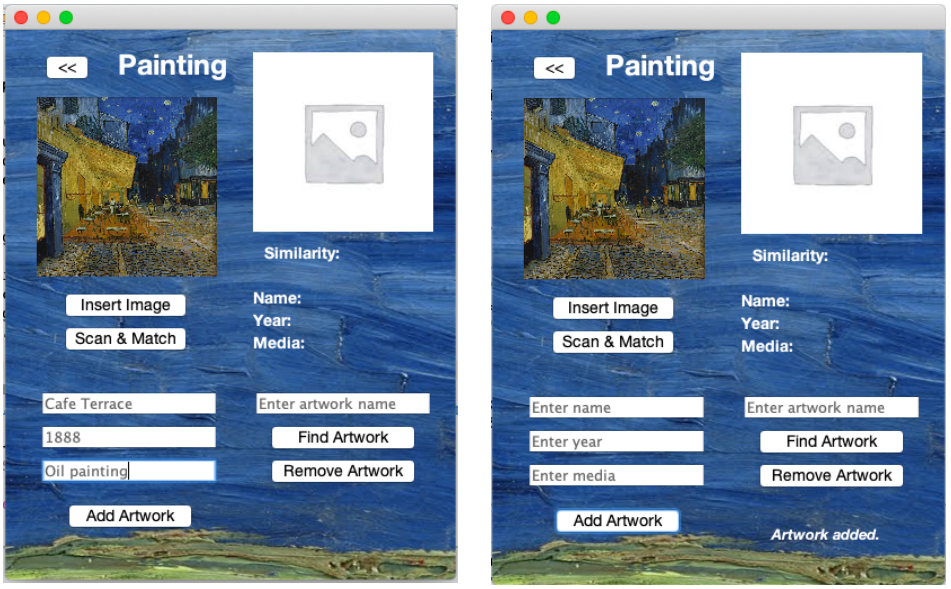
c) No Matched Artwork Evidence

Image 3.5 Result of scanning an image (Cafe Terrace at Night) that is not in the ArrayList.

The image path and information that has been set in the constructor are used. These are the values set at index 0. When there are no matches, the index is set to 0 and the getters will return the values that were set in the constructor.

d) Add Artwork Evidence

As seen in image 3.5, Cafe Terrace at Night has not been inputted into the ArrayList. The following images will demonstrate the addition of the artwork.



Images 3.6 After entering the details and clicking the “Add Artwork” button, the “Artwork added” text is shown on the bottom right.

e) Find Artwork Evidence

The next images demonstrate finding the artwork just from its name.

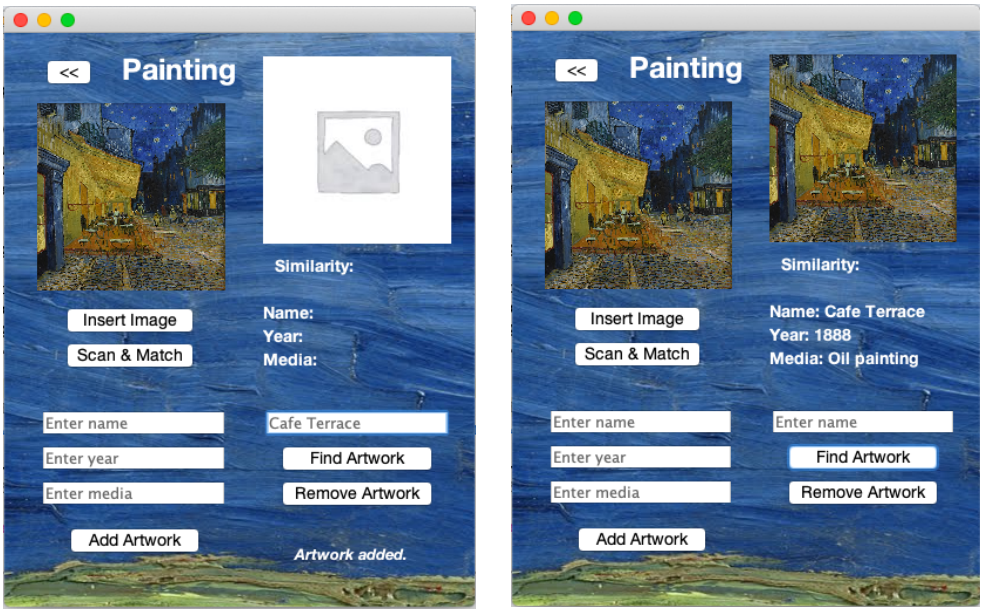
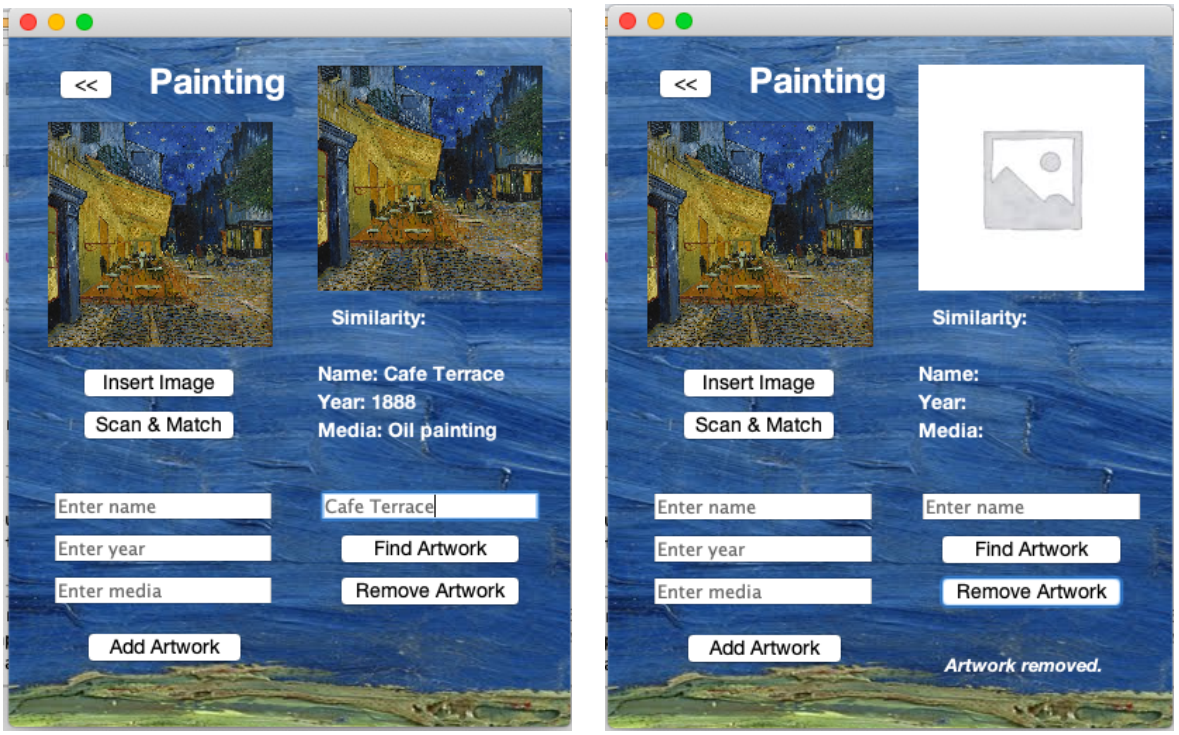


Image 3.7 Entering the name and getting results after pressing the “Find Artwork” button

f) Remove Artwork Evidence



Images 3.8 Removing the Cafe Terrace artwork

Similar to adding an artwork, there is a message at the bottom right of the screen that tells the user what actions were taken.

g) Artwork Not Found Evidence

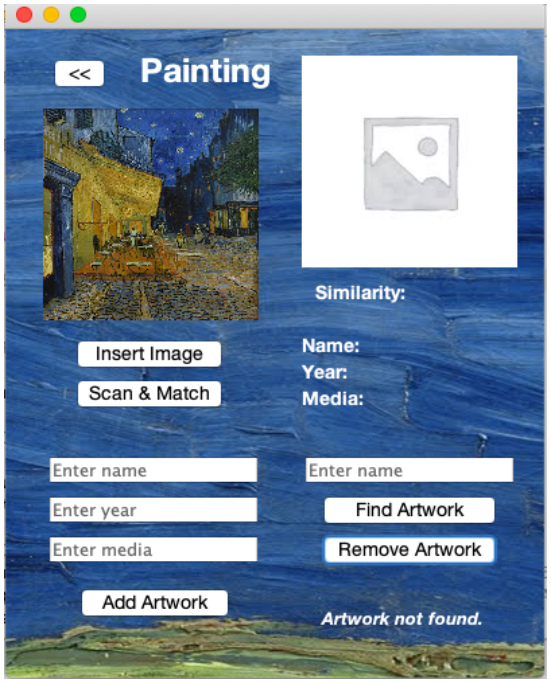


Image 3.9 Artwork not found message

For all cases (adding, removing, and finding), if an artwork is not in the ArrayList, there will be a message at the bottom right saying that the artwork is not found.

4. Resources

GeeksforGeeks. (2022, January 31). *Image processing in java - comparison of two images*. https://www.geeksforgeeks.org/image-processing-in-java-comparision-of-two-images/?ref=lbp

Knowledge to Share. (2018, October 11). *How to open one JFrame from another in java swing : JFrame tutorial* [Video]. YouTube. https://www.youtube.com/watch?v=3dlvseTkRHg

Maurice Muteti. (2019, July 5). *Browse for image file and display it on jlabel using java swing* [Video]. YouTube. https://www.youtube.com/watch?v=p4HV2zDcANI

Wikipedia contributors. (2022a, June 3). *The letters of Vincent van Gogh*. Wikipedia. https://en.wikipedia.org/wiki/The_Letters_of_Vincent_van_Gogh

Wikipedia contributors. (2022b, June 3). *Vincent van Gogh*. Wikipedia. https://en.wikipedia.org/wiki/Vincent_van_Gogh