Image Annotator

Due: 23 November 2020

Kamran Bhatti 3807942

Contents

Brief (Child, 2020)

> *"You are required to develop and document a Java application as follows. The application allows the user to load and view images from files. The user should be presented with a button or similar mechanism to allow them to select an image from the disk. If the selected file cannot be loaded as an image, the user should be informed with a warning message. Once the image is loaded it should be displayed in the UI, with the UI resizing itself to fit the image. The user should be able to click on the image and add an annotation to that particular location on the image. The annotation should indicate the location and have a short text label (see illustrations below). Any number of annotations can be added to the image. The user should be able to click on an existing annotation and remove it. This implies that while adding and removing annotations, the annotations should be drawn as overlays on top of the unchanged image. Finally, the application should allow the annotated image to be saved to a new image file. The saved file should be an ordinary image with the annotations now embedded as a part of the image – reloading it would not allow annotations to be removed after saving."*

Requirements

User stories

End-user	**As an end-user, I want to…**
	• choose a file from my computer to load as an image • annotate an image when I click on the image • remove any annotations that I've added to an image • save the image that I have edited to my computer

Table 1

Table 1 depicts any user stories elicited from the Brief .

Use case diagram

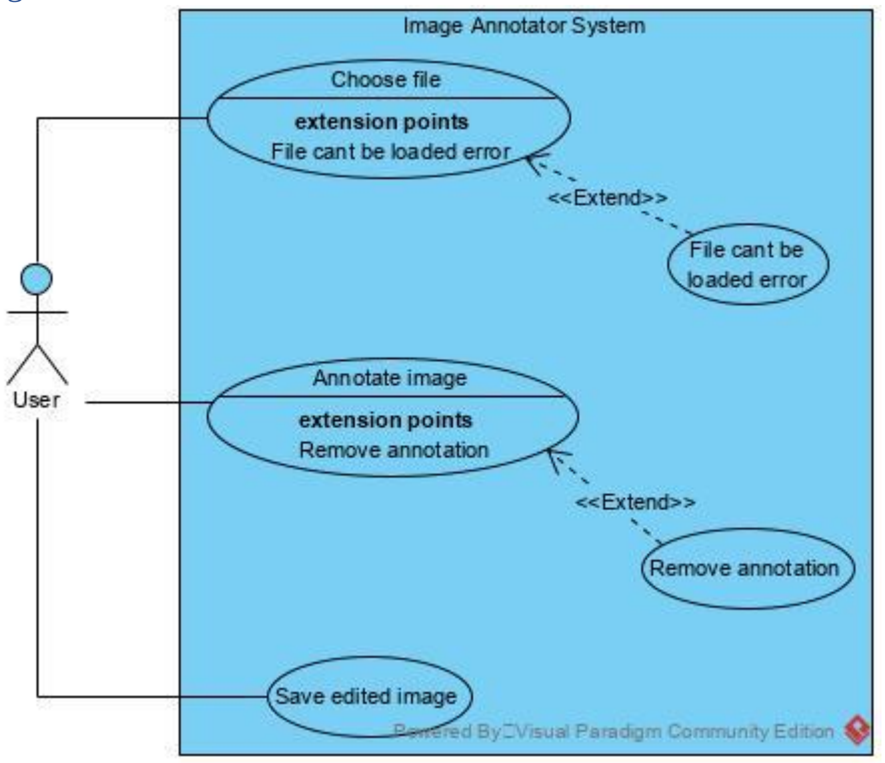


Figure 1

Figure 1 is an illustrated depiction of the user stories expressed in Table 1 using methods depicted in (Phillips & Lemac, 2011). “Choose file” will be executed by a user clicking a button to choose an image on their system via a file explorer window. The “File can't be loaded error” is an extended relationship of “Choose file” as it is only triggered when a file can’t be displayed as an image, to which an error message will be displayed to convey to this error to the user. “Annotate image” would be executed when a user clicks anywhere on the image to which a dialogue box will be displayed asking a user to enter text to annotate onto the image. “Remove annotation” is an extended relationship of “Annotate image” as an annotation must exist before a user can remove an annotation. Executing “Remove annotation” will display a dialogue box asking a user to confirm the deletion of an annotation. “Save edited image” will be executed by a user clicking a save button, a file explorer will be displayed allowing a user to save the edited image to any directory of their choice.

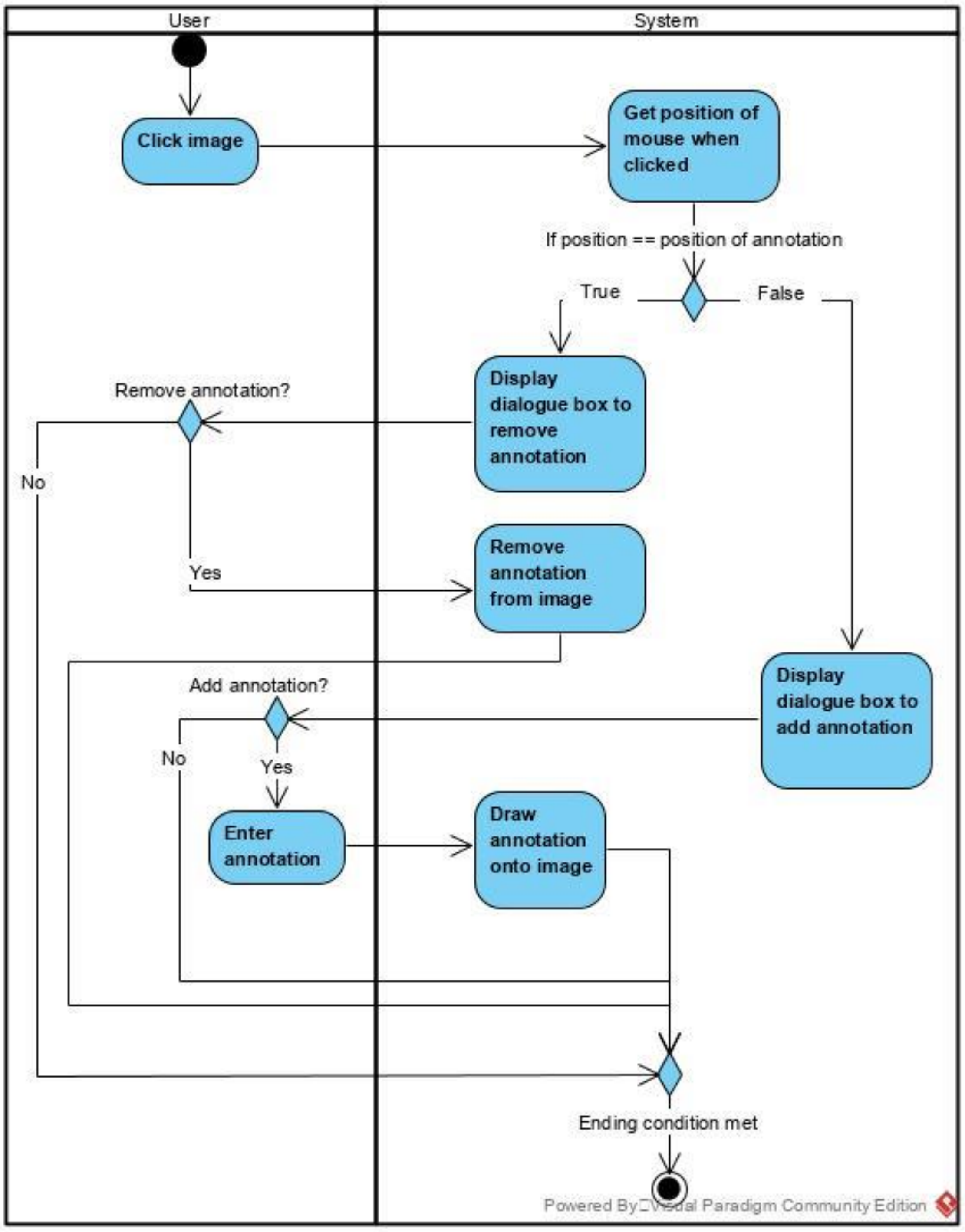
Use case description

Use case name:	Annotate image	
Scenario:	The user wants to add an annotation to the image	
Triggering event:	User clicks the image	
Brief description:	User clicks the image to add an annotation to the image	
Actors:	End-user	
Related use cases:	Choose file Remove annotation	
Stakeholders:		
Preconditions:	Image is loaded in the program	
Postconditions:	Annotation is written onto image	
Flow of activities:	**Actor**	**System**
	1. User clicks on the image	1.1 System gets position where the mouse was clicked 1.2 System displays dialogue box to the user
	2. User enters annotation into a dialogue box	2.1 System takes text input 2.2 System draws annotation on the image
Exception conditions:	1.1 If the position where the user has clicked already has an annotation, the “Remove annotation” use case should be executed instead	

Table 2

Table 2 depicts how the “Annotate image” use case will function. The stakeholders’ section has been omitted as the program isn’t targeted at a specific audience. A check to see if the area clicked has an annotation must be performed before choosing between executing the “Remove annotation” and “Annotate image” use cases. The “Remove annotation” use case will execute when the area clicked is already populated with an annotation while “Annotate image” will execute when the area clicked doesn’t have an annotation.

Activity diagram



User
System
Click image
Get position of mouse when clicked
If position == position of annotation
True
False
Display dialogue box to remove annotation
Remove annotation?
No
Yes
Remove annotation from image
Display dialogue box to add annotation
Add annotation?
No
Yes
Enter annotation
Draw annotation onto image
Ending condition met
Powered By Visual Paradigm Community Edition

Figure 2

(Lemac & Phillips, 2011) was used to create Figure 2 which depicts the events required to trigger the “Annotate image” and “Remove annotation” use cases where “Annotate image” would follow the activities where the position clicked isn’t equal to a position of a previous annotation whilst “Remove annotation” would follow the activities where the position clicked is equal to a position of a previous annotation.

Design

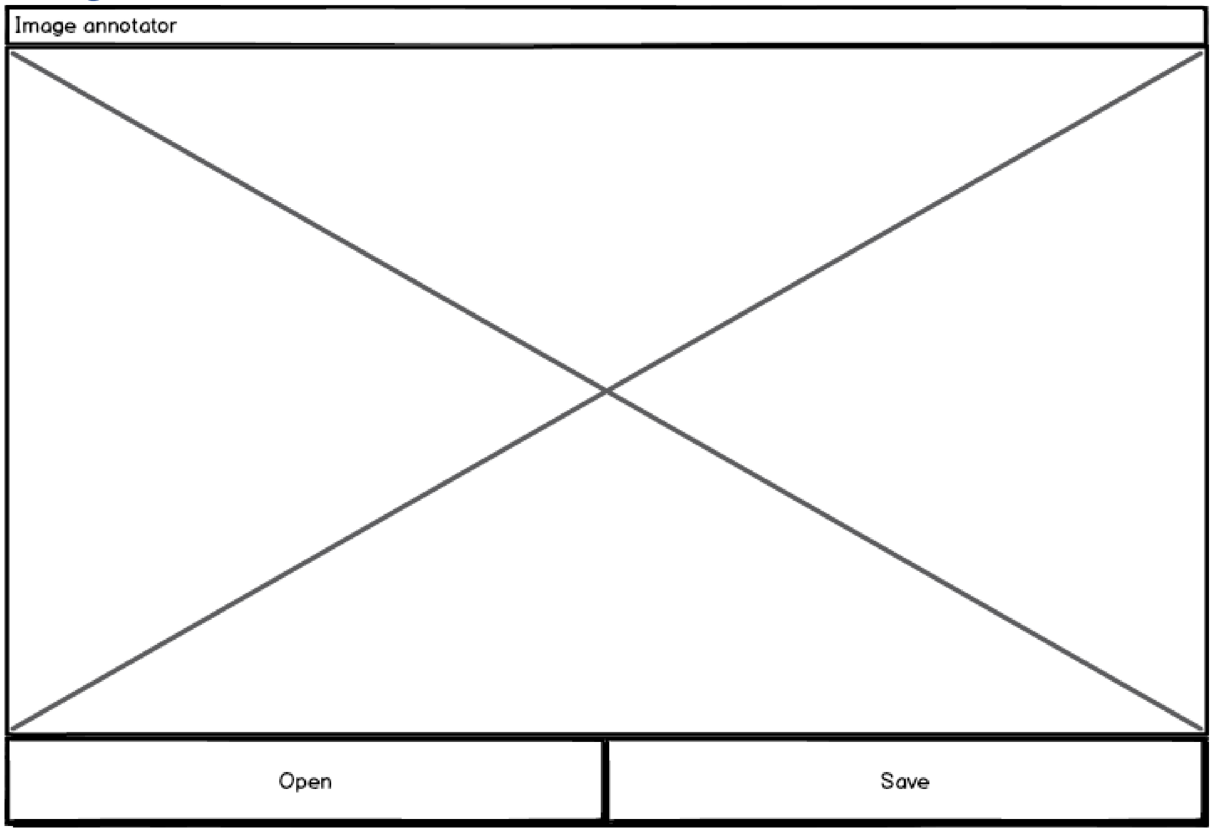


Figure 3

Figure 3 represents how an image would look when a user selects an image from their file explorer after clicking the open button. The application will display a window titled "Image annotator" with an open and save button, to open an image and save an annotated image. The space represented by an "X" is a placeholder for an image. When the open button is clicked a file explorer window will be displayed to allow a user to select an image to open. Similarly, the save button will display a file explorer window when clicked however, this will allow an annotated image to be saved to the users' file system.

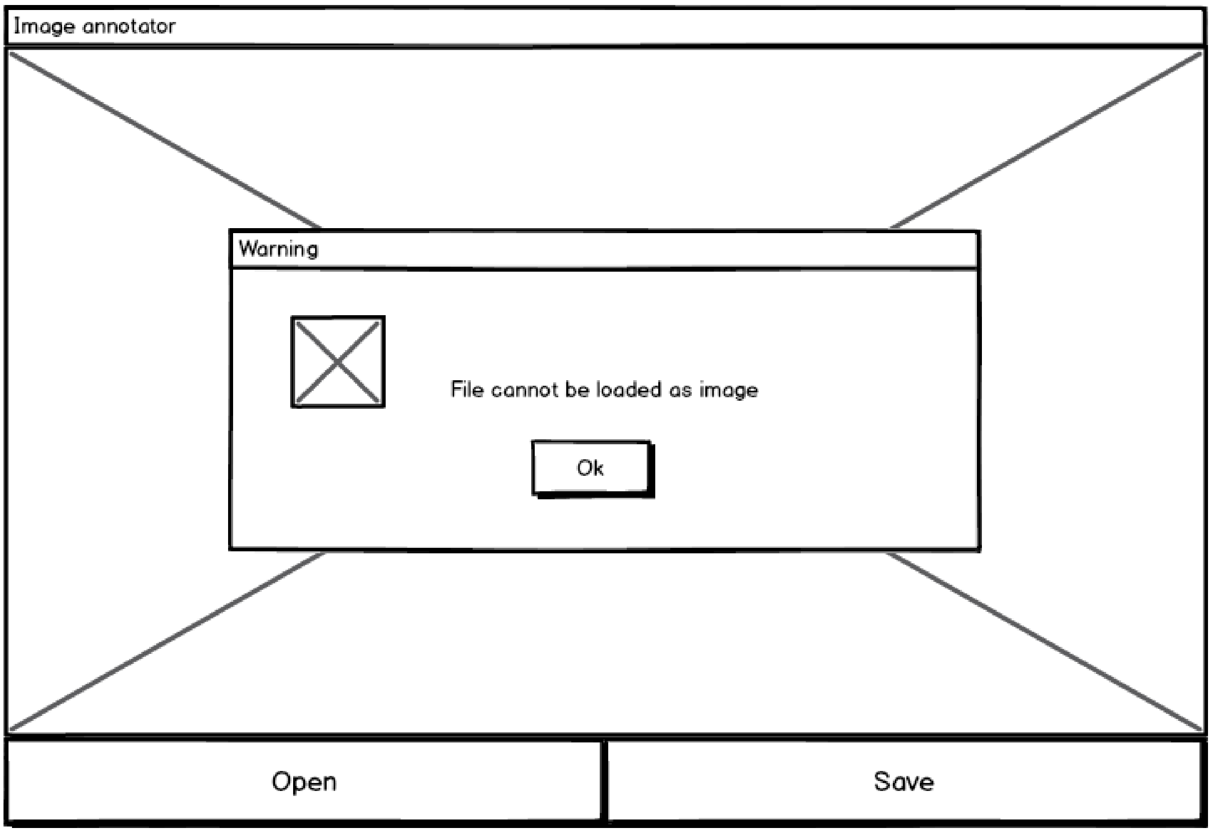


Figure 4

Figure 4 represents a warning message which would be displayed if a user chooses to a file which can’t be loaded as an image.

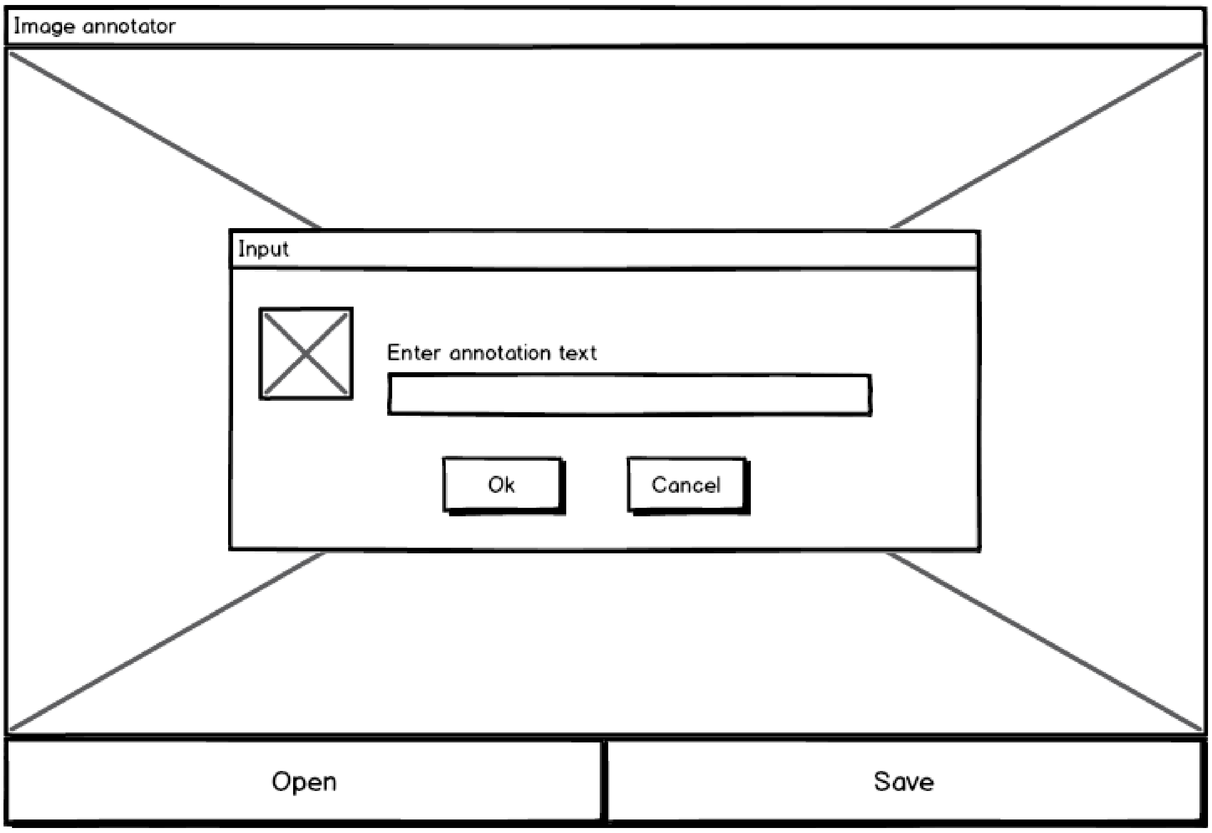


Figure 5

If the user clicks on the image, a dialogue box will be displayed to annotate the image as shown in Figure 5. If a user enters text into the text input field and clicks “ok”, the image will be annotated with the specified text, else no annotation will be added to the image.

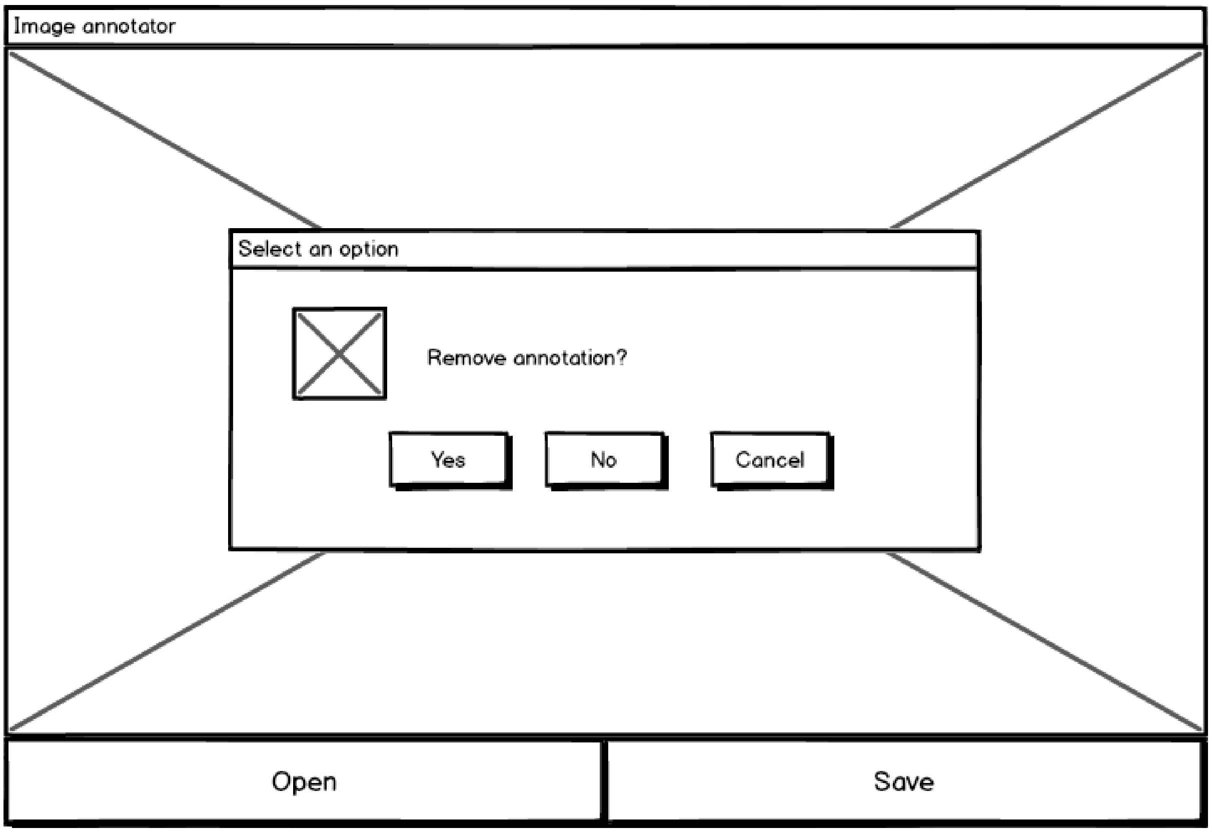


Figure 6

If the user selects the text they have annotated, they are greeted with a dialogue box asking them if they want to remove the annotation as shown in Figure 6.

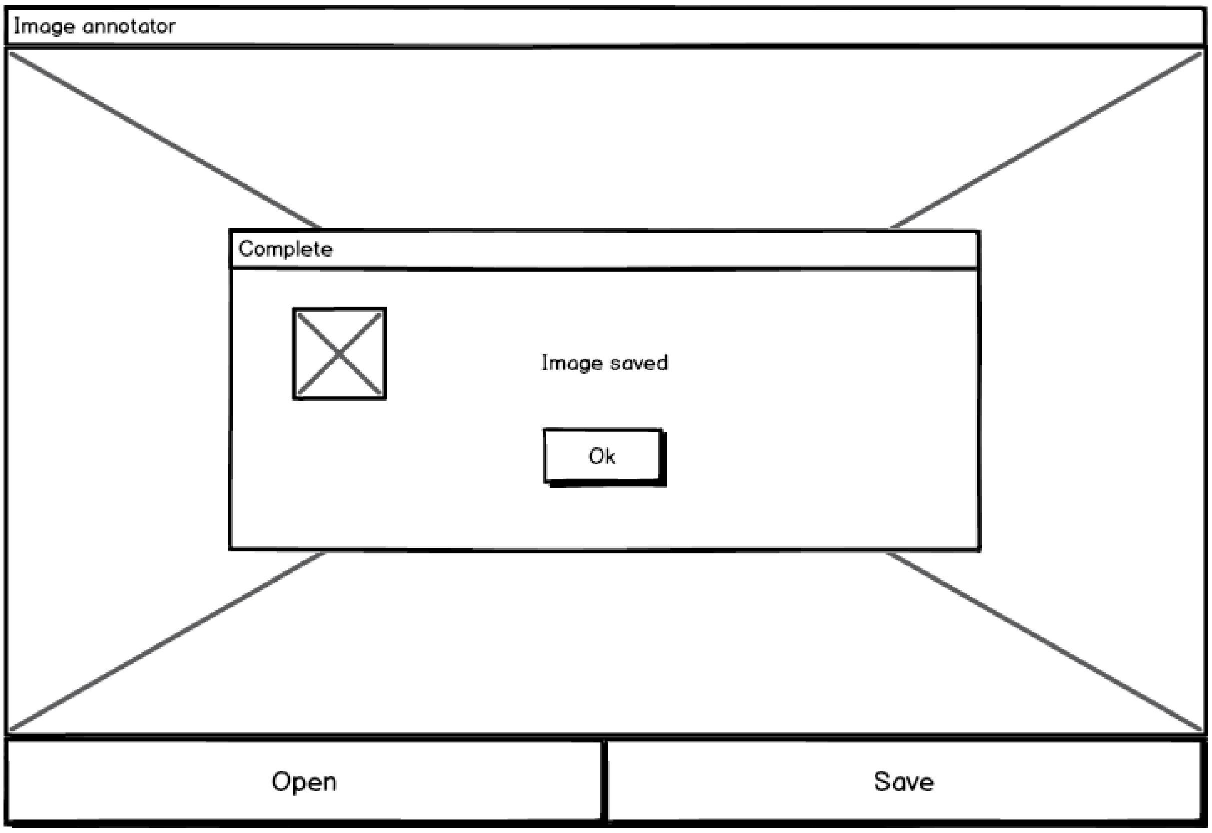
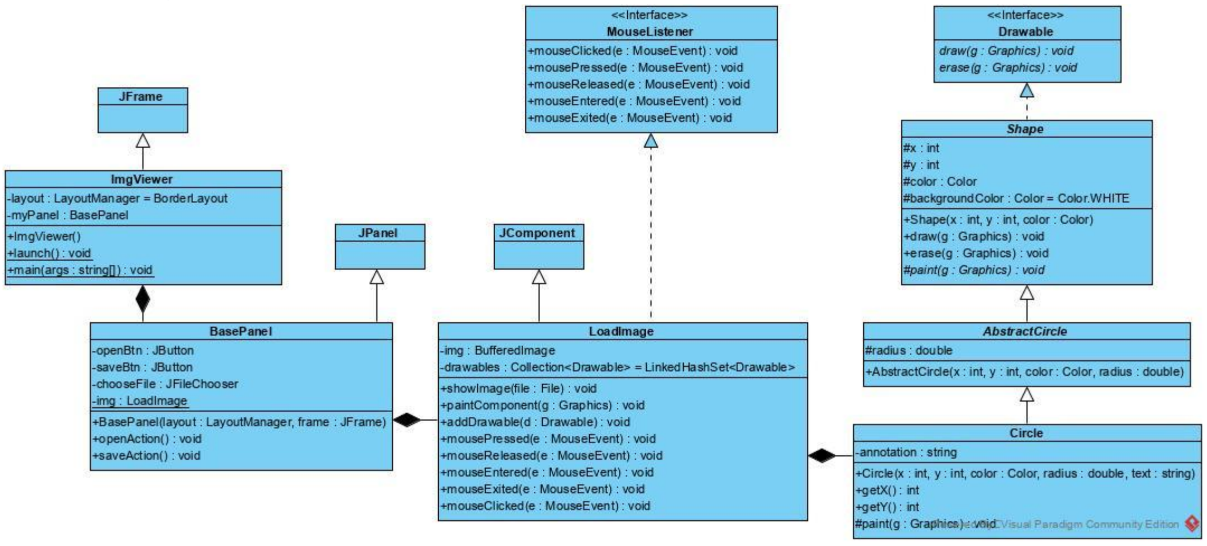


Figure 7

If the user clicked the save button to save the annotated image, a dialogue box that the image has been saved will be displayed as shown in Figure 7.

Class diagram



<<Interface>>
MouseListener
+mouseClicked(e : MouseEvent) : void
+mousePressed(e : MouseEvent) : void
+mouseReleased(e : MouseEvent) : void
+mouseEntered(e : MouseEvent) : void
+mouseExited(e : MouseEvent) : void
<<Interface>>
Drawable
draw(g : Graphics) : void
erase(g : Graphics) : void
JFrame
Shape
#x : int
#y : int
#color : Color
#backgroundColor : Color = Color.WHITE
+Shape(x : int, y : int, color : Color)
+draw(g : Graphics) : void
+erase(g : Graphics) : void
#paint(g : Graphics) : void
ImgViewer
-layout : LayoutManager = BorderLayout
-myPanel : BasePanel
+ImgViewer()
+launch() : void
+main(args : string[]) : void
JPanel
JComponent
BasePanel
-openBtn : JButton
-saveBtn : JButton
-chooseFile : JFileChooser
-img : LoadImage
+BasePanel(layout : LayoutManager, frame : JFrame)
+openAction() : void
+saveAction() : void
LoadImage
-img : BufferedImage
-drawables : Collection<Drawable> = LinkedHashSet<Drawable>
+showImage(file : File) : void
+paintComponent(g : Graphics) : void
+addDrawable(d : Drawable) : void
+mousePressed(e : MouseEvent) : void
+mouseReleased(e : MouseEvent) : void
+mouseEntered(e : MouseEvent) : void
+mouseExited(e : MouseEvent) : void
+mouseClicked(e : MouseEvent) : void
AbstractCircle
#radius : double
+AbstractCircle(x : int, y : int, color : Color, radius : double)
Circle
-annotation : string
+Circle(x : int, y : int, color : Color, radius : double, text : string)
+getX() : int
+getY() : int
#paint(g : Graphics) : void
Visual Paradigm Community Edition

Figure 8

The “launch” and “main” methods in the “imgViewer” class are underlined as they are static methods which are similar to the “img” variable in the “BasePanel” class which is a static “LoadImage” instance. In the “Drawable” interface, the methods are abstract but have unspecified visibility thus an access modifier doesn’t classify the usage of the interfaces’ methods. Furthermore, the “paint” method in both the “Shape” and “Circle” classes are protected methods with a void return type but, in the “Shape” class the “paint” method is abstract. Lastly, composition relationships have been used to describe how each class is related to each other for example, a “BasePanel” instance is instantiated in the “ImgViwer” class while an instance of “LoadImage” would be instantiated in the “BasePanel” class.

Implementation

All source code can be found at (Bhatti, 2020).

ImgViewer class

```
public class ImgViewer extends JFrame {


    public ImgViewer() throws HeadlessException {
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setTitle("Image annotator");

        LayoutManager layout = new BorderLayout();
        BasePanel myPanel = new BasePanel(layout, this);
        getContentPane().add(myPanel);

        pack();
        setVisible(true);
    }

    public static void launch() {
        new ImgViewer();
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> launch());
    }


}
```

Figure 9

Code from (Child, 2006) was used as a reference to use the “JFrame” component in Figure 9. In the class constructor, an instance of “BasePanel” is initialized with a “BorderLayout” which is then added to the “JFrame” thus, all components in the “BasePanel” will be displayed in the “JFrame”.

BasePanel class

```
private JButton openBtn = new JButton("Open");

private JButton saveBtn = new JButton("Save");

private JFileChooser chooseFile = new JFileChooser();

private static LoadImage img = new LoadImage();
```

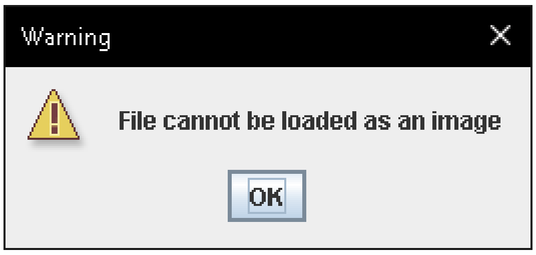
Figure 10

Figure 10 depicts the attributes inside the class. “JButton”s are used for the open and save button whilst a “JFileChooser” is used for the “chooseFile” variable. The “img” variable instantiates a “LoadImage” object, however, it uses a static classifier as depicted in (srisar, 2009) so that when a new image is opened in the application, the image being displayed is replaced with the image a user chooses.

```
public BasePanel(LayoutManager layout, JFrame frame) {

        super(layout);

        this.openBtn.addActionListener((ev) -> {

            try {

                openAction();

                frame.pack();

            } catch (Exception e) {

                JOptionPane.showMessageDialog(frame, "File cannot be loaded as an image", 
"Warning", JOptionPane.WARNING_MESSAGE);

            }

        });

        this.saveBtn.addActionListener((ev) -> {

            try {

                saveAction();

            } catch (IOException ex) {

                Logger.getLogger(BasePanel.class.getName()).log(Level.SEVERE, null, ex);

            }

        });

        JPanel BtnPanel = new JPanel(new GridLayout(1, 2));

        BtnPanel.add(this.openBtn);

        BtnPanel.add(this.saveBtn);

        add(BtnPanel, BorderLayout.SOUTH);

}
```

Figure 11

Figure 11 is the constructor in the “BasePanel” class which takes a “LayoutManager” and “JFrame” variable as arguments when called. The “LayoutManager” variable is passed to the superclass, “JPanel”, to change the layout for the panel to the layout specified when the class is instantiated as “BasePanel” extends “JPanel”. Similar to (Child, 2006), action listeners were used to link a button to a method when the button is clicked. The buttons are enclosed in try-catch blocks as they link to methods which manipulate files thus if during the execution of a method an error is encountered the code in the catch section of the try-catch block will be executed. For example, when the open button is clicked, an “openAction” method is executed where a user can select a file to load as an image. If the file selected is an image, “frame.pack()” is executed which will resize the “JFrame” to fit all the components in the “JPanel” but if the file can’t load as an image, (Oracle, 1995) has been used to render an error message that the image couldn’t be displayed as shown in Figure 12. (Eels, 2014) details how to move the buttons to the bottom of the “JPanel” to which a “JPanel” is created using a “GridLayout” where each button is added to that “JPanel”. After each button is added to the button “JPanel”, the button “JPanel” is added to the “BasePanel”’s’ “JPanel” at the bottom of the window.


Warning

File cannot be loaded as an image

Figure 12

```
public void openAction() throws IOException {

        int returnVal = this.chooseFile.showDialog(BasePanel.this, "Select an image");

        if (returnVal == JFileChooser.APPROVE_OPTION) {

            File file = this.chooseFile.getSelectedFile();

            this.img.showImage(file);

            add(this.img, BorderLayout.CENTER);

        }

        this.chooseFile.setSelectedFile(null);

        // resets the selected file so that the file chooser variable can be reused

}
```

Figure 13

“openAction” is executed when the open button is clicked. (Oracle, 1995) is used to implement the “JFileChooser” variable so that when the open button is clicked, a file chooser is displayed asking the user to select an image. If the user chooses an image, the file is passed to the “LoadImage” class via the “img” variable and then added to the main “JPanel” in the centre of the window. After the image has been added to the centre of the “JPanel”, the selected file is reset via the “setSelectFile” method executed on the “chooseFile” variable.

```
public void saveAction() throws IOException {

        int returnVal = this.chooseFile.showSaveDialog(BasePanel.this);

        if (returnVal == JFileChooser.APPROVE_OPTION) {

            File file = this.chooseFile.getSelectedFile();

            BufferedImage image = new BufferedImage(this.img.getWidth(), this.img.getHeight(),
BufferedImage.TYPE_INT_RGB);

            Graphics g = image.createGraphics();

            this.img.printAll(g);

            g.dispose();

            ImageIO.write(image, "jpg", file);

            JOptionPane.showMessageDialog(this, "Image saved", "Complete",
JOptionPane.INFORMATION_MESSAGE);

        }

        this.chooseFile.setSelectedFile(null);

}
```

Figure 14

“saveAction” is executed when the save button is clicked. (Oracle, 1995) is used to display a file chooser to save the edited image as a new image file. If the user saves the file as a new image, methods in (MadProgrammer, 2013) are used to write all of the contents from the “JComponent” into an image file. This is executed by using the “Graphics” class in Java to write the image and any annotations on the image to a “BufferedImage” variable which is written to the users' file system using the” ImageIO” class as detailed in (Oracle, 1995). The JPG extension is used as the identifier of the image as the JFIF file loaded into the program is classified as a form of JPG. (Oracle, 1995) was used to provide feedback to a user after the image is saved to the users' file system in the form of a message box as shown in Figure 15. After the image is saved the “chooseFile” variable is reset.

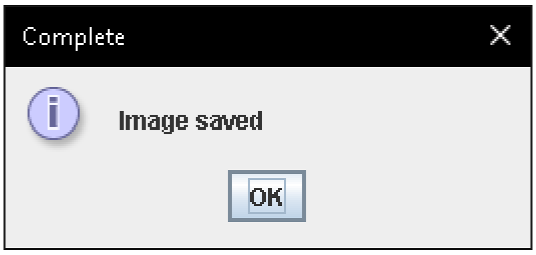

Complete

Image saved

Figure 15

LoadImage class

```
private BufferedImage img;

private final Collection<Drawable> drawables = new LinkedHashSet<Drawable>();
```

Figure 16

Figure 16 depicts attributes in the "LoadImage" class where "img" is a "BufferedImage" variable while "drawables" is a collection of "Drawable" objects implemented as a "LinkedHashSet" as depicted in (Child, 2020).

```
public void showImage(File file) throws IOException {

        removeMouseListener(this);

        this.drawables.clear();

        this.img = ImageIO.read(file);

        this.setPreferredSize(new Dimension(this.img.getWidth(this), this.img.getHeight(this)));

        addMouseListener(this);

        this.repaint();

}
```

Figure 17

"showImage" is executed when opening an image as the "LoadImage" class doesn't have a constructor. The method takes a file as an argument when called to which (Oracle, 1995) is used to clear all values in the "drawables" HashSet as when a new image is loaded into the program, all annotations from the previous image should be removed from the programs memory. (Oracle, 1995) is used to read the image passed as a file to "ImageIO" while (Child, 2006) was used to set the preferred size of the image by passing the dimensions of the loaded image to the "setPreferredSize" method of the "JComponent" extended by the "LoadImage" class. As depicted in (Oracle, 1995), a mouse listener has been added to the image to locate where a user clicks on the image. "this.repaint()" is executed so that the image is updated is a new image is opened. An error continued to occur in which loading a new image would trigger the "mouseListener" multiple times according to the number of images loaded in the program, to remedy this, the "removeMouseListener" function is used to reset the "mouseListener" object being used for each image.

```
    public void addDrawable(Drawable d) {
        this.drawables.add(d);
        repaint();
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g = g.create();
        if (this.img != null) {
            g.drawImage(this.img, 0, 0, this);
            g.setFont(getFont().deriveFont(12f));
            for (Drawable d : this.drawables) {
                d.draw(g);
            }
        }
    }
```

Figure 18

(Child, 2020) is used in Figure 18 to add “Drawable” objects to the “drawables” HashSet in the “addDrawable” method and is also used to draw each “Drawable” object on the loaded image in the “paintComponent” method. The “repaint” method is called in the “addDrawable” method so that when a “Drawable” object is added to the “drawables” HashSet, the “Drawable” object is then drawn on top of the image. (srisar, 2009) is used to draw the image in the “JComponent” using the “drawImage” method executed on the “Graphics” variable “g”. (Child, 2006) is used to set the font of each annotation a user makes before the annotation is drawn on top of the image, however, instead of using font size 24, font size 12 is used so that the annotations aren’t too large.

```
    public void mousePressed(MouseEvent e) {

    }


    public void mouseReleased(MouseEvent e) {

    }


    public void mouseEntered(MouseEvent e) {

    }


    public void mouseExited(MouseEvent e) {

    }


    public void mouseClicked(MouseEvent e) {

        int count = 0;

        Drawable annotation = new Circle(0, 0, Color.WHITE, 0, "");

        for (Drawable d : this.drawables) {

            if ((((Circle) d).getX() - 10 < e.getX()) && (((Circle) d).getX() + 10 > e.getX()) &&
(((Circle) d).getY() - 10 < e.getY()) && (((Circle) d).getY() + 10 > e.getY())) {

                count = 1;

                annotation = d;

            }

        }

        if (count != 1) {

            String text = (String) JOptionPane.showInputDialog(this, "Enter annotation text", "Input",
JOptionPane.QUESTION_MESSAGE);

            if (text != null && text.length() > 0) {

                this.addDrawable(new Circle(e.getX(), e.getY(), Color.YELLOW, 10, text));

            }

        } else {

            int returnVal = JOptionPane.showConfirmDialog(this, "Remove annotation?");

            if (returnVal == JOptionPane.YES_OPTION) {

                this.drawables.remove(annotation);

                repaint();

            }

        }

    }
```

Figure 19

As the “LoadImage” class implements the “MouseListener” interface, the “mousePressed”, “mouseReleased”, “mouseEntered”, “mouseExited” and “mouseClicked” methods must be present in the class however, they do not need to have any code in their methods. “mouseClicked” has executable code to register where a user wants to annotate on an image. A for loop is used to iterate over each of the “Drawable” objects in the “drawables” HashSet to check if the position of an annotation on the image is similar to the position of the mouse when a user clicks on the image. This is implemented by calling the “getX” and “getY” methods of each “Circle” object and comparing the position of the object with the position of the mouse. If the user clicks inside the circle where an annotation is drawn on the image, the “count” variable is changed and the annotation is stored in a “Drawable” variable called “annotation”.

A conditional statement is used to check if the “count” variable has changed. If the “count” variable is 0, a dialogue box as depicted in (Oracle, 1995) is displayed asking a user to enter an annotation where the entered text is stored in a “String” variable named “text”. If the “text” variable contains any inputted data, a “Circle” object is instantiated with the position of the mouse when the image was clicked, a Color to make the circle yellow when drawn, a radius of 10 pixels and the text entered by the user. This “Circle” object is added to the “drawables” HashSet via the “addDrawable” method which is they drawn on top of the image. Figure 20 illustrates the dialogue box that would be displayed when a user clicks on the image if the area clicked doesn’t have an annotation.

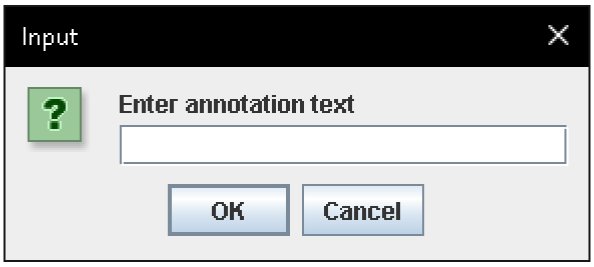


Figure 20

If the count variable is 1, a dialogue box as seen in (Oracle, 1995) is displayed which asks a user if they want to remove the annotation drawn. If the user chooses to remove the annotation, the “annotation” variable storing the “Drawable” object which was clicked is removed from the “drawables” HashSet via the “remove” method. To update the displayed image, the “repaint” method is executed which redraws all of the “Drawable” objects that are left in the “drawables” HashSet. Figure 21 depicts the dialogue box that would be displayed is a user clicks on an annotation.

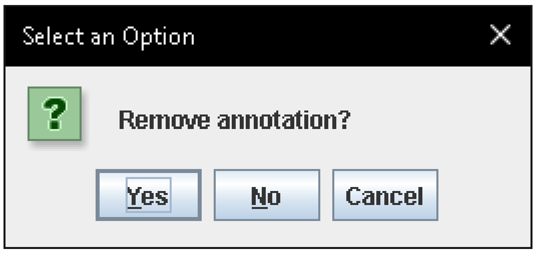


Figure 21

Circle class

```
public class Circle extends AbstractCircle {


    private String annotation;


    public Circle(int x, int y, Color color, double radius, String text) {

        super(x, y, color, radius);

        this.annotation = text;

    }


    public int getX() {

        return x;

    }


    public int getY() {

        return y;

    }


    protected void paint(Graphics g) {

        g.drawOval((int) (x - radius), (int) (y - radius), (int) (2 * radius), (int) (2 * radius));

        g.drawString(this.annotation, x + 20, y + 20);

    }

}
```

Figure 22

The “Circle” class utilises the “AbstractCircle” class, “Shape” class and “Drawable” interface found in (Child, 2020). However, this class has a “String” attribute named “annotation” which stores the “text” a user enters in the annotation dialogue box for each “Circle” object via “this.annotation”. The “getX” and “getY” methods are used to return the coordinates of the “Circle” object for each annotation. Inside the “paint” method, a circle is drawn using the “drawOval” method on the “Graphics” variable “g” which is also used to draw the “annotation” variable with the “drawstring” method as seen in (Child, 2006) thus each circle drawn is drawn with an annotation near it.

MVC architecture

Utilising an MVC architecture would separate code between a model, view and controller which would improve the overall organisation of the application and would allow for easier application maintenance. MVC is more appropriate when implementing large scale systems which require long term support if the interface were changed the overall functionality of the system would remain the same as the model would remain unchanged but the connections between the view classes and model classes facilitated through the controller classes may remain the same. However, for simple applications utilising the MVC architecture is not necessary as it can make simplistic actions quite complex. For instance, if a user wanted to update a text label on an interface, the view class would call a method in a controller class which calls a method in a model class to update the label found in the view class which could have been simplified to directly updating the label in the class the variable is found in if MVC isn't used.

To format the current code into an MVC architecture, the classes which extend "JFrame", "JPanel" and "JComponent" would be placed in the view package thus "ImgViewer", "LoadImage" and "BasePanel" would be placed in the view package. However, the "launch" and "main" methods in the "ImgViewer" class would go in a controller package. The "launch" method would call the "ImgViewer" class in the view package and a class in the model package. The lambda expressions in "BasePanel" would call methods in the controller package. "openAction" and "saveAction" would most likely be in the controller class but as these methods create "JFileChooser" objects, the "openAction" and "saveAction" methods would remain in the "BasePanel" class with the "JFileChooser" objects but the operation of writing the edited image to a new image file would occur in a different class. Therefore, "openAction" and "saveAction" would remain in the "BasePanel" class but the code to write the currently edited image to a new image file would be found in the model package where a class in the controller package references the method to execute the writing of an image thus connecting the "BasePanel" class in the view package to a class in the model package via a class in the controller package. The methods in the "LoadImage" class would be unaffected as all methods in the class directly change components on the user interface.

The "Shape" class, "Circle" class, "AbstractCircle" class and "Drawable" interface would be treated as classes in the model package, however, the "draw" and "erase" methods in the "Shape" class and the "paint" method in the "Circle" class would go in a class in the view package as they directly affect the components on the interface. Therefore, when these methods need to be executed from a class in the model package calling directly to the class methods in the view package.

Appraisal

I believe that the application conforms to the design specifications detailed in the Brief as users can load images, annotate images, remove annotations and save annotated images. Also, a warning message as seen in Figure 12 and dialogue boxes depicted in Figure 20 and Figure 21 have been implemented as dictated by the Brief whilst, an additional dialogue box has been implemented to improve the usability of the application by providing feedback to a user that the process of saving an image has been completed as shown in Figure 15.

However, the user must select the circle associated with each annotation instead of clicking on the annotated text drawn onto the image which seems counter-intuitive as if a user wants to remove an annotation, it would make more sense to click on the annotated string to remove the annotation instead of the area on the image the annotation was assigned to. Furthermore, if the window is resized, the size of the image won't resize to the size of the window which causes annotation dialogue boxes to appear when clicking on a space that the image doesn't occupy in a resized window. To remedy this, you could use the code specified in Figure 23 which would resize the image according to the dimensions of the current window by calling "getWidth" and "getHeight" methods, however, annotations added to the image won't update their positions as they store their position according to where the mouse was clicked on the image for the images' previous dimensions.

```
g.drawImage(this.img, 0, 0, getWidth(), getHeight(), this);
```

Figure 23

Although an MVC architecture is impractical for simplistic applications, if it were used when developing the application, future maintenance of the system would be easier. Other developers could modify the system without referring to specialized documentation or without checking all the classes as they could modify the view, model or class packages.

Bibliography

Bhatti, K., 2020. *ImgViewer.* [Online]
Available at: https://github.com/k5924/ImgViewer
[Accessed 22 October 2020].

Child, M., 2006. *05 - Java GUIs.* [Online]
Available at: https://vle.lsbu.ac.uk/mod/resource/view.php?id=1454662
[Accessed 22 October 2020].

Child, M., 2006. *06 - Custom GUI components and MVC.* [Online]
Available at: https://vle.lsbu.ac.uk/mod/resource/view.php?id=1454664
[Accessed 28 October 2020].

Child, M., 2020. *04 - Abstract classes, interfaces and UML.* [Online]
Available at: https://vle.lsbu.ac.uk/mod/resource/view.php?id=1454660
[Accessed 3 November 2020].

Child, M., 2020. *Coursework Assignment 1.* [Online]
Available at: https://vle.lsbu.ac.uk/mod/resource/view.php?id=1508111
[Accessed 22 October 2020].

Eels, H. F. O., 2014. *Java GUI Moving buttons to bottom of the page.* [Online]
Available at: https://stackoverflow.com/questions/27198796/java-gui-moving-buttons-to-bottom-of-the-page
[Accessed 25 October 2020].

Lemac, M. & Phillips, N., 2011. *Week 5: Activity Diagrams.* [Online]
Available at:
https://vle.lsbu.ac.uk/pluginfile.php/1725334/mod_resource/content/3/ActivityDiagrams.pptx
[Accessed 22 March 2020].

MadProgrammer, 2013. *Exporting a JPanel to an image.* [Online]
Available at: https://stackoverflow.com/questions/17690275/exporting-a-jpanel-to-an-image
[Accessed 3 November 2020].

Oracle, 1995. *Class LinkedHashSet<E>.* [Online]
Available at: https://docs.oracle.com/javase/7/docs/api/java/util/LinkedHashSet.html
[Accessed 3 November 2020].

Oracle, 1995. *How to Make Dialogs.* [Online]
Available at: https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html
[Accessed 1 November 2020].

Oracle, 1995. *How to Use File Choosers.* [Online]
Available at: https://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html
[Accessed 26 October 2020].

Oracle, 1995. *How to Write a Mouse Listener.* [Online]
Available at: https://docs.oracle.com/javase/tutorial/uiswing/events/mouselistener.html
[Accessed 3 November 2020].

Oracle, 1995. *Writing/Saving an Image.* [Online]
Available at: https://docs.oracle.com/javase/tutorial/2d/images/saveimage.html
[Accessed 3 November 2020].

Phillips, N. & Lemac, M., 2011. *Week 4: Use Case Descriptions.* [Online]
Available at:
https://vle.lsbu.ac.uk/pluginfile.php/1725329/mod_resource/content/5/UseCaseDescriptions.pptx
[Accessed 17 February 2020].

srisar, 2009. *How to display image in a panel with JFileChooser.* [Online]
Available at: https://www.java-forums.org/awt-swing/21973-how-display-image-panel-jfilechooser-print.html
[Accessed 27 October 2020].