

Written Report

May 11, 2007

“Paint-A-Bit” PNG Image Editing Program

Kevin Alvarado

Submitted in partial fulfillment  
Of the requirements for CSCI 490  
Senior Project

## Table of Contents

Table of Contents .....	2
Table of Figures .....	3
Introduction.....	4
Introduction.....	4
Significance.....	4
Project Specifications .....	4
Background .....	4
Functional Specifications .....	4
User Interface .....	4
Technical Details.....	6
Architectural Design .....	7
Class Inheritance Diagrams.....	9
File Data Structure .....	11
Summery.....	12
Glosary.....	13
Reference .....	14

## Table of Figures

Figure 1 - Model-View-Controller relationship .....	6
Figure 2 - Class Diagrams .....	7
Figure 3 - Class Diagrams continued .....	8
Figure 4 - ImageArea Inheritance .....	9
Figure 5 - ColorSliderPanel & ColorSwatch Inheritance.....	9
Figure 6 - PaletteArea Inheritance .....	10
Figure 7 - EditorFrame Inheritance.....	10

## **Introduction**

“Paint A Bit” is an image editing program used to create and modify portable network graphics (png) images that are to be optimized for mobile phone application development. The application will offer limited drawing functionality such as drawing lines, circles, boxes and color fills. The application will also offer means to alter colors and color order within the image.

## **Significance**

There are limited resources available when developing software for mobile phones. This being the case, it is beneficial to reduce the amount of resources consumed wherever possible. “Paint A Bit” will be used to remove unnecessary data contained within the png file format, reducing the file size but maintaining image integrity.

There are other commercial applications available that already manage the creating and modification functionality I am proposing. However, it has been my experience that these commercial applications lack the functionality to optimize the image data for mobile phone applications.

## **Project Specifications**

The main goal of “Paint A Bit” project is to allow the user to create and edit png image files optimized for mobile phone software applications. This will be accomplished with the use a graphical user interface that allows the user to alter the image and adjust color data associated with that image.

## **Background**

Mobile devices have limited resources available for software. Unlike a desktop system, it is often impractical to add memory or upgrade a mobile devices processor. This being the case, application developers must always consider optimizations for every aspect of the application. The png image format consists of various data blocks that store meta data regarding the file, such as creation time, color information, image data, text data, histogram information, chromatic data, and other information. Although, this meta data may be relevant for image archival information, most data blocks are not required to display the image on a mobile device.

## **Functional Specifications**

Functionality can be broken down into three categories, file, color, and image operations. File operations will allow the user to load and save image and palette files. Color operations will allow the user to select and alter individual colors contained within the palette by altering a colors red, green and blue color values. Image operations will allow the user to alter the image with simple drawing operations, such as drawing lines, circles, squares and color fills.

## **User Interface**

The user interface will utilize Java Swing components, such as JMenu, JMenu Item, JButton, JSlider, JToolBar, JFileChooser and others to be identified as needed. The menus of the project will use JMenu and JMenuItem components and will organize the program commands into the three functionality categories identified above. A JToolBar component will give the user quick access to the most used commands, to be identified as the project is developed.

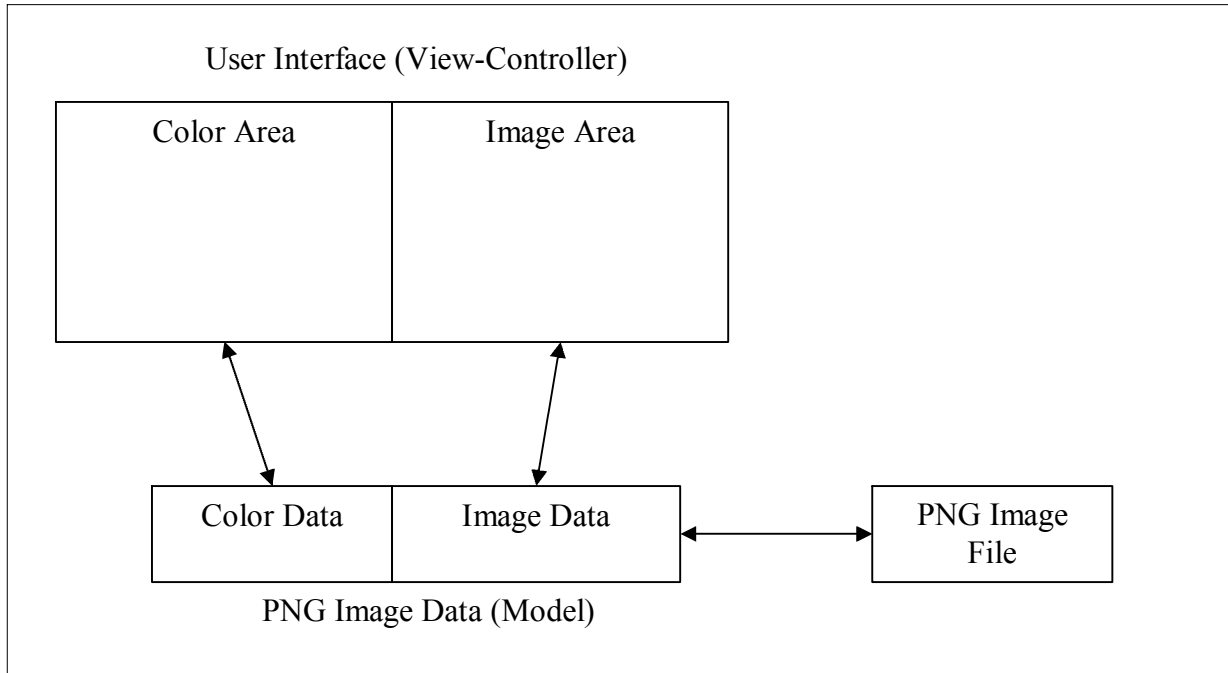
The main portion of the application will consist of the “work area” which will be divided into two parts. One part will display the color data of the image called the “palette editor”; the other will display the image called the “image editor”.

The palette editor will consist of a 16x16 grid of color swatch. The color swatches will represent the palette data stored within the image and will allow the user to select a color to edit by a mouse click on the grid. Below the color swatch grid will be three JSlider and three JTextField components used to edit the selected swatch’s red-green-blue values.

The image editor portion of the application will use a JScrollPane used to display the image. The JScrollPane is needed to allow the image to be scrolled within the edit area when the user selects a zoom level that causes the image screen size to exceed the work area’s screen size. Also contained within the image editor area will be a JToolBar used as a “drawing tool” palette, allowing the user to select from the four basic drawing tools, lines, circles, rectangles and color fills.

## Technical Details

“Paint A Bit” will use the Model-View-Controller design. The user interface consisting of Java Swing components will provide the “View-Controller” portion, while the “Model” will consist of the png image data. The project will only allow for the viewing/controlling of a single model (image) at any point in time. The model can be divided into two sub-models, color data and image data. Both sub-models will be reflected in the “work area’s” sub-views, “palette editor” and “image editor.” See diagram 1 below.



**Figure 1 - Model-View-Controller relationship**

The png image data (model) will consist of an object-oriented hierarchy of classes. Initially the hierarchy will be three classes, Color, Palette, and Image. The relation between the classes is shown in diagram 2.

## Architectural Design

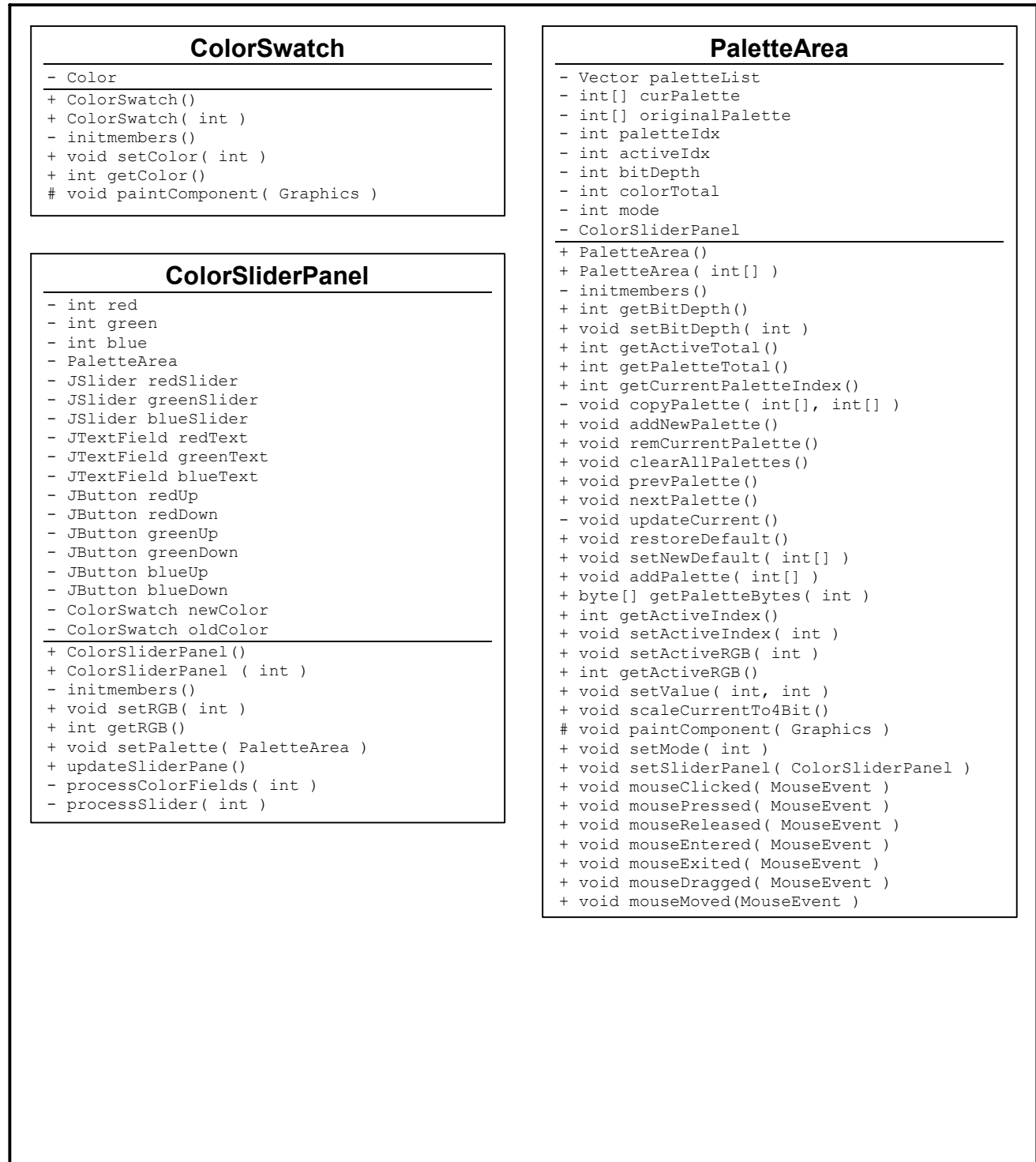


Figure 2 - Class Diagrams

### MultiPalettePNG

```
- int colorCount
- int bitDepth
- int[] paletteData
- int imageWidth
- int imageHeight
- int[] imageData
- boolean transparent
- int paletteCount
- Vector paletteList
+ initMembers()
+ MultiPalettePNG()
+ MultiPalettePNG( String )
+ int getColorCount()
+ int getBitDepth()
+ int getImageWidth()
+ int getImageHeight()
+ boolean hasTransparent()
+ int getPaletteCount()
+ int[] getPalette()
+ int[] getPalette( int )
+ int[] getImageData()
+ int[][] getImageRoster()
+ int[][] getMultiPalette()
+ void setTransparent( boolean )
+ boolean setImageData( int[] )
+ boolean readImageData( String )
- int getBlockType( byte[] )
- int bytesToInt( byte[] )
- byte[] intToBytes( int )
- byte[] readDataBlock( FileInputStream )
- boolean readPNGHeader( FileInputStream )
- boolean processPNG_IHDR( byte[] )
- boolean processPNG_PLTE( byte[] )
- boolean processPNG_tRNS( byte[] )
- boolean processPNG_IDAT( byte[] )
- boolean processPNG_IEND()
- boolean processPNG_mPLT( byte[] )
+ boolean writeImageData( String )
```

### ImageArea

```
- int activeTool
- int startX
- int startY
- int stopX
- int stopY
- int width
- int height
- int scale
- BufferedImage bufferedImage
- IndexColorModel colorModel
- WritableRaster writeRaster
- Graphics2D g2D
- MultiPalettePNG mpngImage
+ ImageArea( MultiPalettePNG )
- void resetSize()
+ void setImageScale( int )
+ void setActiveTool( int )
+ void setActiveColor( int )
+ void mouseClicked( MouseEvent )
+ void mousePressed( MouseEvent )
+ void mouseReleased( MouseEvent )
+ void mouseEntered( MouseEvent )
+ void mouseExited( MouseEvent )
+ void mouseDragged( MouseEvent )
+ void mouseMoved( MouseEvent )
```

### EditorFrame

```
- JSplitPane editorImageSplitPane
- JPanel palettePanel
- PaletteArea palette
- JLabel indexIndicatorLabel
- ColorSliderPanel sliders
- JPanel imagePanel
- ImageArea imageArea
- MultiPalettePNG mpImage
- ImageIcon displayImage
+ EditorFrame()
- void fillPaletteValues()
- byte[] intToBytes( int )
- void loadImageFile()
- void saveImageFile()
- void closeOnExit()
- void recolorImage()
- void addNewPalette()
- void deleteCurrentPalette()
- void prevPalette()
- void nextPalette()
- void restoreDefaultPalette()
- void make4BitColor()
- File getFileReference( int )
```

Figure 3 - Class Diagrams continued

## Class Inheritance Diagrams

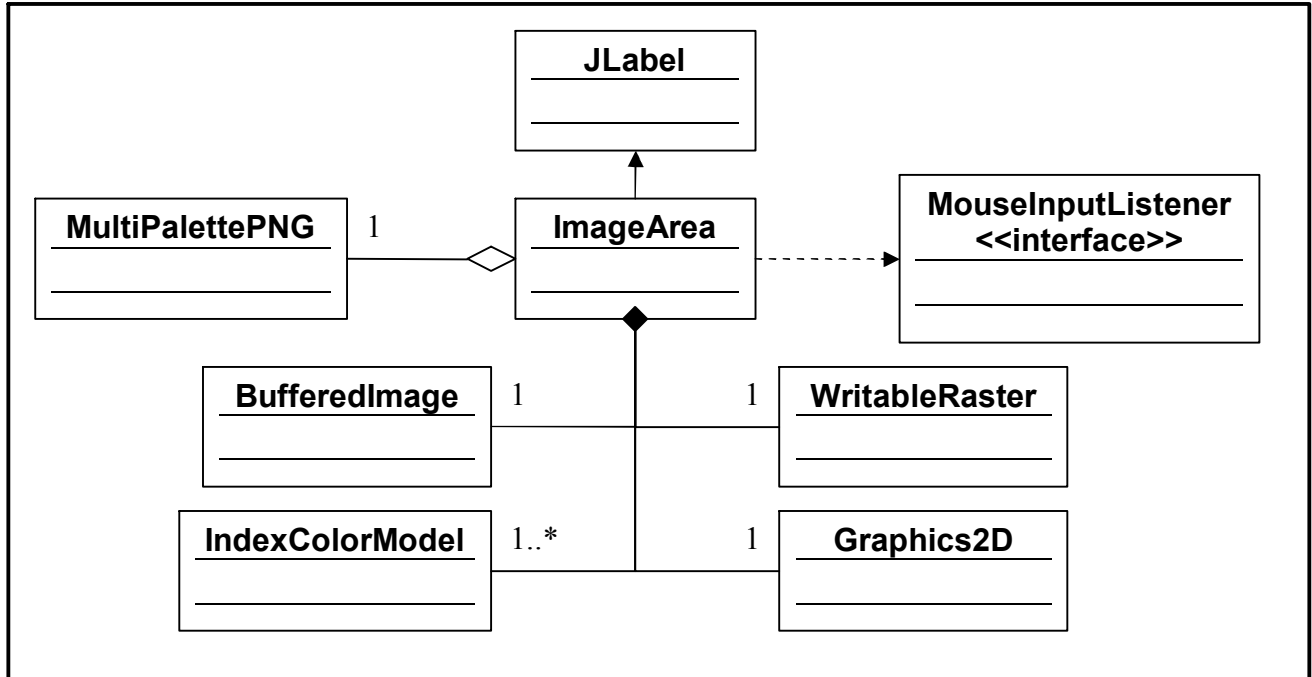


Figure 4 - ImageArea Inheritance

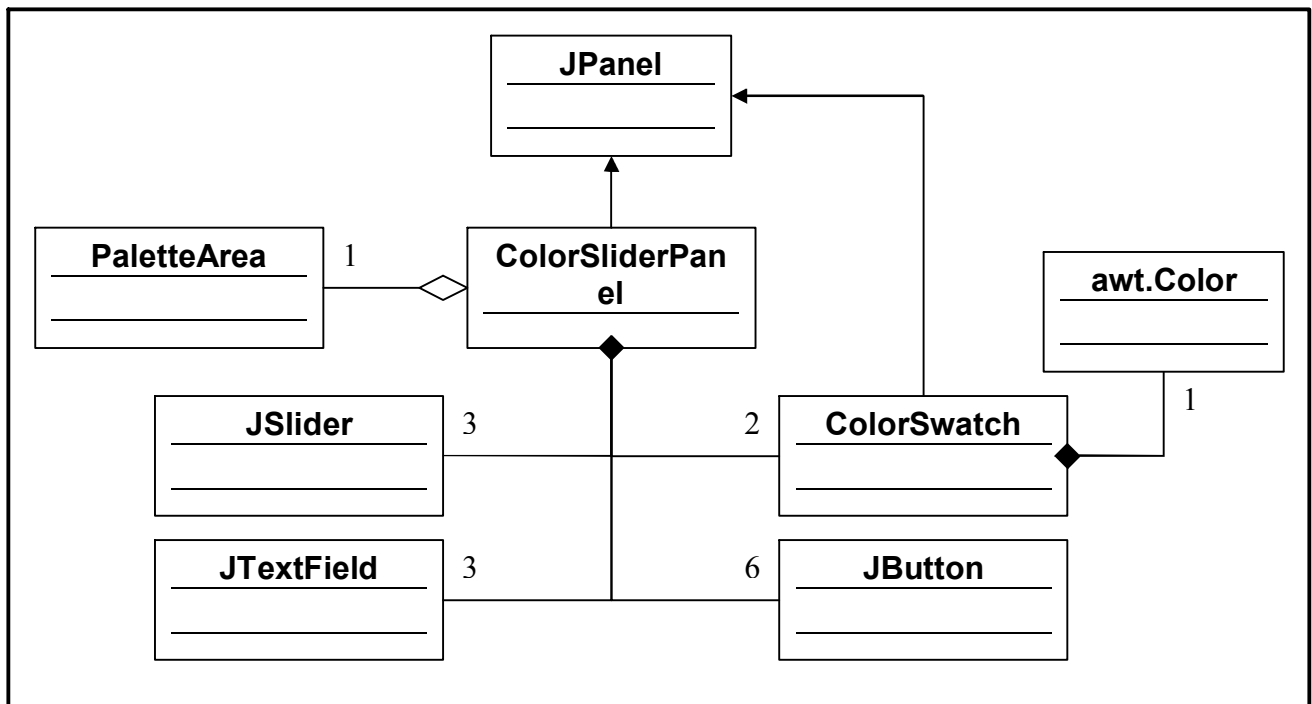


Figure 5 - ColorSliderPanel & ColorSwatch Inheritance

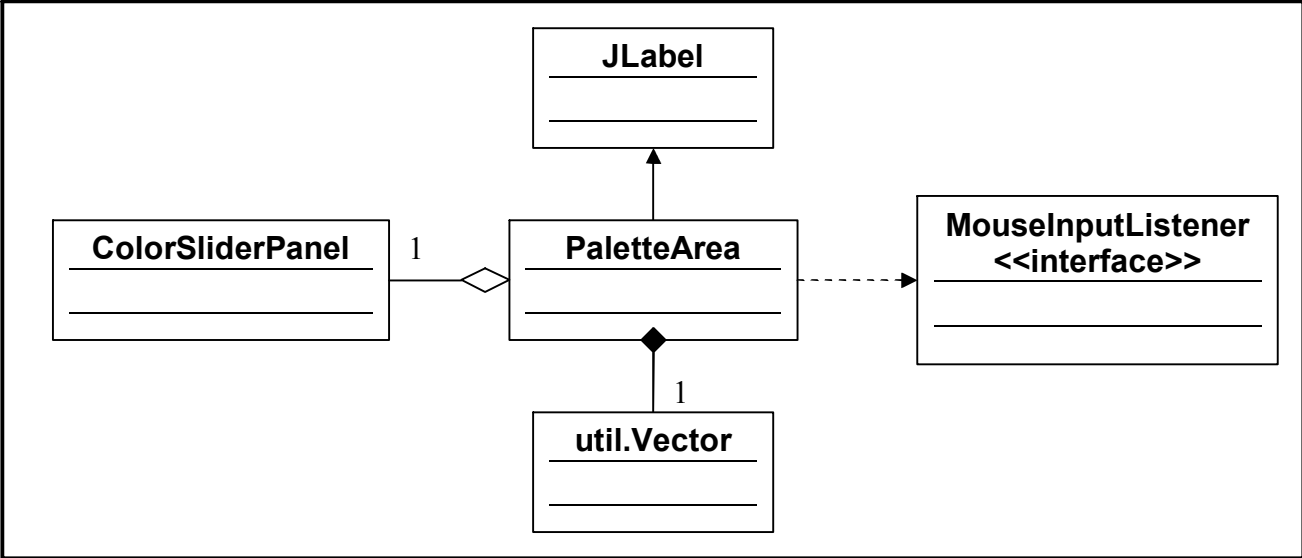


Figure 6 - PaletteArea Inheritance

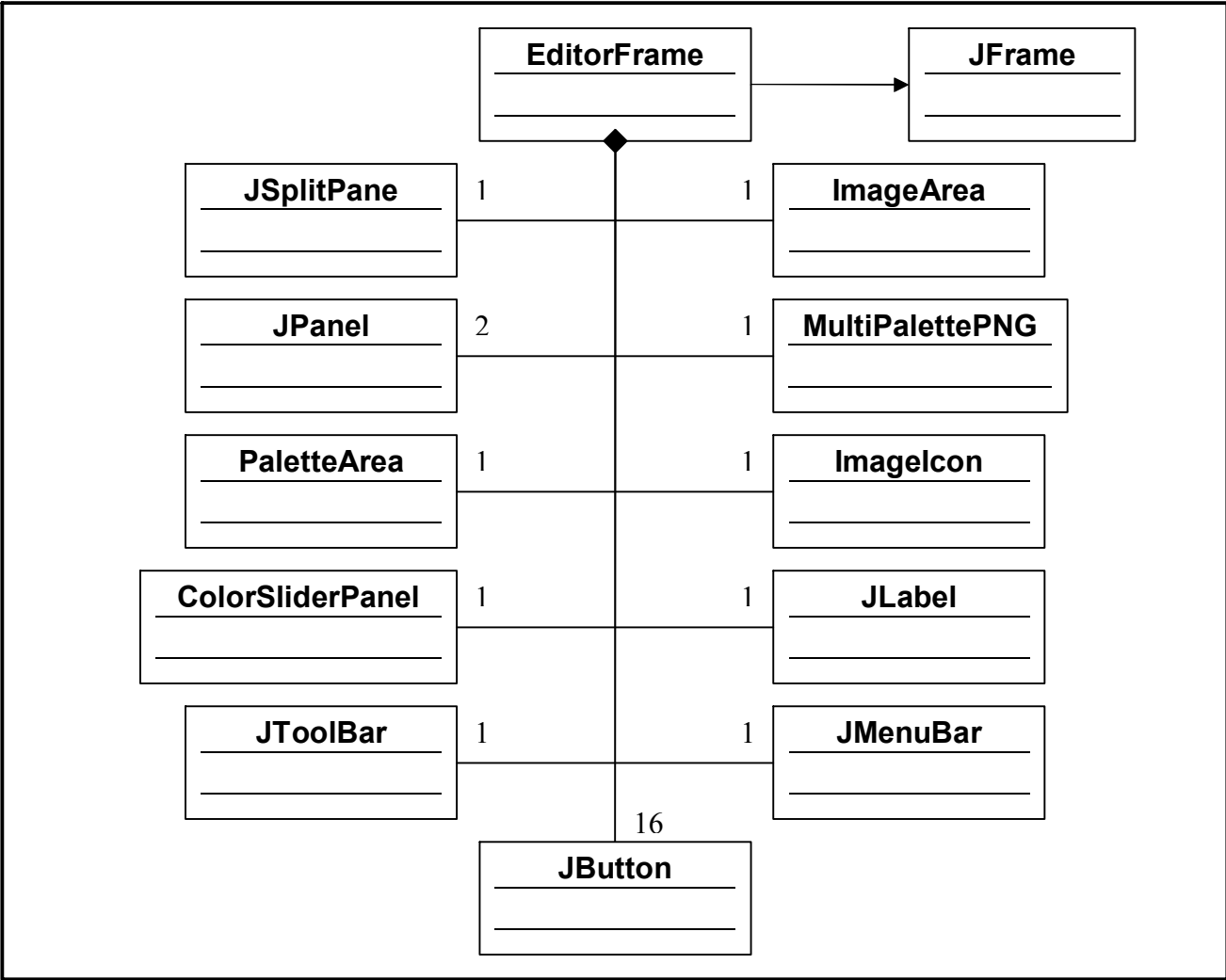


Figure 7 - EditorFrame Inheritance

## File Data Structure

The data structure used by “Paint-A-Bit” will be the PNG file format [PNG] defined by the specification published on the web by the World Wide Web Consortium [W3C].

The file format defined for png images has a required order of defined data blocks. The “Paint-A-Bit” program will adhere to this standard for reading image data from files. “Paint-A-Bit” program will also write image data to files using the defined standard [PNG], with the addition to the mPLT block.

Type	Size	
PNG File Identifier	8 bytes	0x89 0x50 0x4E 0x47 0x0D 0x0A 0x1A 0x0A
IHDR block size	4 bytes	value of 0x0000000D
IHDR identifier	4 bytes	0x49 0x48 0x44 0x52
IHDR block data	12 bytes	Contains image meta data, width, height, depth, etc.
IHDR block CRC	4 bytes	CRC generated code from IHDR identifier & block data
PLTE block size	4 bytes	pSize = 3 bytes * colors used
PLTE identifier	4 bytes	0x50 0x4C 0x54 0x45
PLTE block data	pSize bytes	Contains rgb bytes for each color entry
PLTE block CRC	4 bytes	CRC generated code from PLTE identifier & block data
tRNS block size	4 bytes	tSize = number of colors with alpha values
tRNS identifier	4 bytes	0x74 0x52 0x4E 0x53
tRNS block data	tSize bytes	Typically 1 byte, indicating color index 0 is transparent.
tRNS block CRC	4 bytes	CRC generated code from tRNS identifier & block data
IDAT block size	4 bytes	iSize = number of bytes used in compressed data stream
IDAT identifier	4 bytes	0x49 0x44 0x41 0x54
IDAT block data	iSize bytes	Image data stored in ZLIB deflate compression stream
IDAT block CRC	4 bytes	CRC generated code from IDAT identifier & block data
mPLT block size	4 bytes	mSize = palette count * ( pSize + 12 )
mPLT identifier	4 bytes	0x6D 0x50 0x4C 0x54
mPLT block data	mSize bytes	1 to palette count number of complete PLTE blocks
mPLT block CRC	4 bytes	CRC generated code from mPLT identifier & block data
IEND block size	4 bytes	IEND blocks have 0 size so value is 0x00000000
IEND identifier	4 bytes	0x49 0x45 0x4E 0x44
IEND block CRC	4 bytes	CRC generated code from IEND identifier. typically 0xAE 0x42 0x60 0x82

\* tRNS and mPLT blocks are optional and may be excluded from file data when not defined.

**Table 1 - PNG File data format**

## Summery

I was the sole contributor to the “Paint-A-Bit” project. During the course of the project I had researched the specifications and file format of the PNG image. Within the course of this research I learned that the PNG image also uses the ZLIB compression for the compression of image data. The success of the project would require proper usage of the ZLIB compression and decompression algorithm. Upon my efforts in construction my own methods to implement the ZLIB algorithms; I discovered that Java had included a utility package, “java.util.zip”, that would implement the ZLIB compression/decompression algorithm.

Also while developing the project I had also discovered within the Java API a set of classes, found under “java.awt.image” package, that I was able to utilize for drawing and creating and changing an image roster. The classes were BufferedImage, IndexColorModel, and WritableRaster. I had used these classes to abstract a layer between my data model of the image and color data to produce the interactive view used in the user interface.

Once all the components were constructed, putting the project together was not difficult. I had taken advantage of the packages that Java and offered to implement the interaction with the user interface (view) and the data model of the multi-palette png image (mode).

## Glosary

<b>Term</b>	<b>Definition</b>
Active Color	Currently selected color to be used for all drawing operations on the image.
Active Tool	Current drawing function to be implemented upon mouse click within the image.
Color Entry	Three values that represent an RGB triple used to produce a color in an image.
Color Swatch	Graphical component within the GUI filled with a color defined by the RGB values for a palette entry.
CRC	Cyclic Redundancy Code. An algorithm designed to detect transmission errors.
Drawing Palette	See <b>Tool Palette</b> .
Image Raster	A collection of palette index values that represent one color per pixel within the image.
Palette	A collection of color entries used by an Image. A palette must have at least two but no more than 256 entries.
Palette Entry	See <b>Color Entry</b> .
Palette Index	The integer value of a color entry's index within an image's palette.
PNG	Portable Network Graphics – an image specification defined by International Standard, ISO/IEC 15948:2003
RGB triple	Three values that represent Red, Green, and Blue quantities of light to produce color on a computer screen.
Tool Palette	A toolbar containing icon buttons that represent drawing functions, such as Lines, Circles, Boxes, etc.
W3C	World Wide Web Consortium ( <a href="http://www.w3c.org">www.w3c.org</a> )

## Reference

[W3C] The applicable W3C standards are published on the web at [www.w3c.org](http://www.w3c.org).

[SUN] The applicable Java libraries and conventions are published through Sun Microsystems, Inc., and are available on the web at [java.sun.com](http://java.sun.com).

[PNG] Portable Network Graphics (PNG) Specification (Second Edition) are published on the web at [www.w3.org/TR/PNG/](http://www.w3.org/TR/PNG/)

[RFC-1950] ZLIB Compressed Data Format Specification version 3.3 defined on the web at <http://www.ietf.org/rfc/rfc1950.txt>

[RFC-1951] DEFLATE Compressed Data Format Specification version 1.3 defined on the web at <http://www.ietf.org/rfc/rfc1951.txt>