Graphic Helper Files

Introduction

In this unit we will be creating two new files to help us work with graphics in the future.

Loading Image Review:

We will load images using the below code:

```
try
{
   img1 = ImageIO.read((new File("Image1Location")));
   img2 = ImageIO.read((new File("Image2Location ")));
}
catch(Exception e)
{
   System.out.println("Error Loading Images: " + e.getMessage());
}
```

The code attempts to load 2 images. If it fails you will see an error message

Buffered Image Overview:

Attributes:

Name	Description
public static final Attributes:	
TYPE_INT_ARGB	RGB color with transparency
TYPE_INT_RGB	RGB with no transparency

Constructor:

Constructor	Description
BufferedImage(int width, int height, int imageType)	Creates a bufferedImage based on the received specifications. Width – The width of the image Height – The height of the image imageType – the graphics mode of the image imageTypes: TYPE_INT_ARGB – RGB color with transparency TYPE_INT_RGB – RGB with no transparency

Methods:

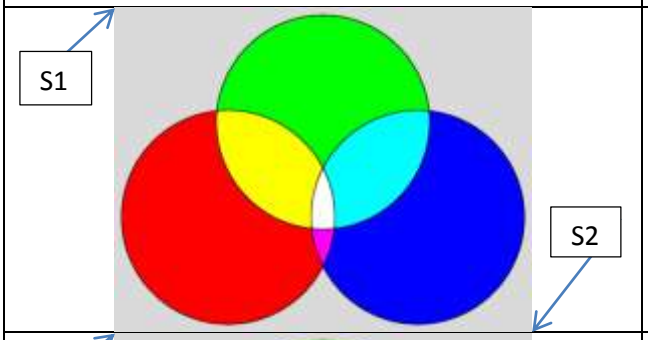
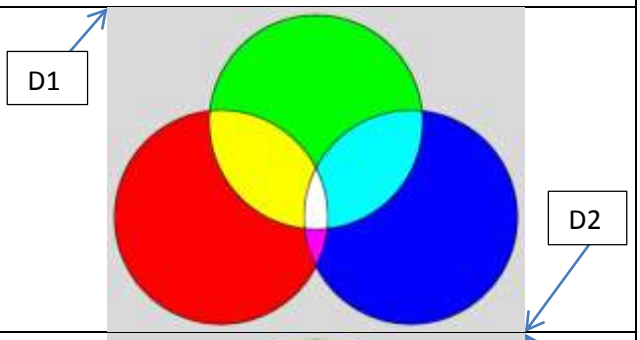
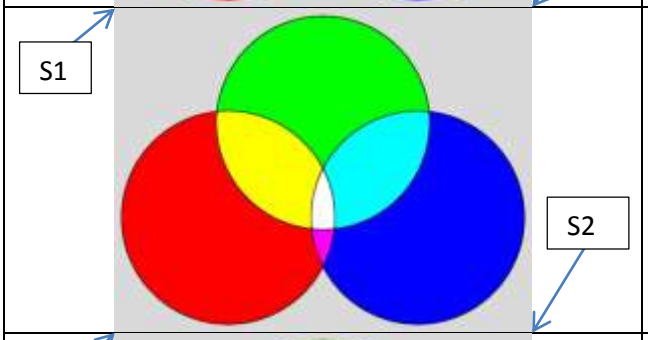
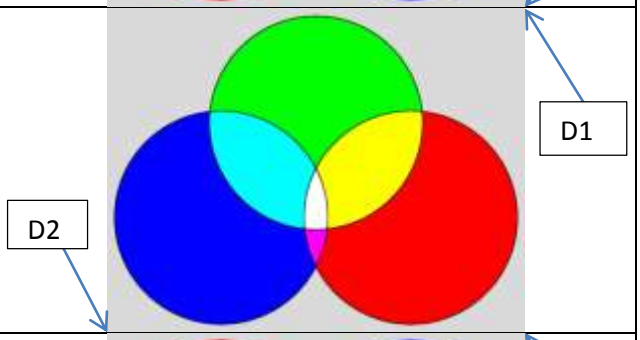
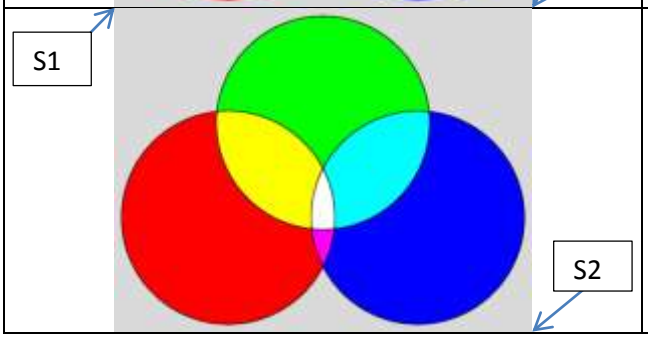
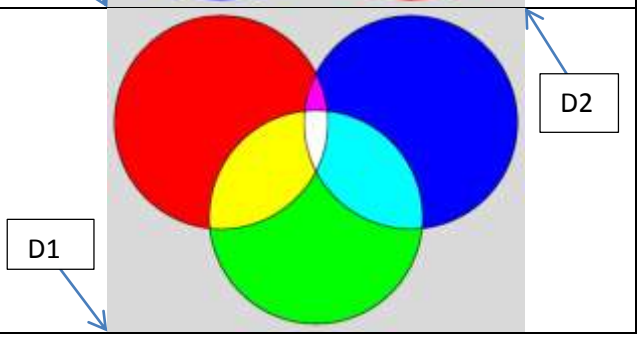
Return Type	Name	Description
ColorModel	getColorModel()	Returns the color model of the image. We will use the color model to get the transparency type of the image. **Example:** int transparency = img.getColorModel().getTransparency(); **The two values for transparency:** Transparency.BITMASK Transparency.TRANSLUCENT
int	getWidth()	Returns the width of the image
int	getHeight()	Returns the height of the image
Graphics	getGraphics()	Returns the graphics of the image
int	getRGB(int x, int y)	Returns the int value of the color at (x, y)
void	setRGB(int x, int y, int rgb)	Changes the color at (x, y) to rgb

New Graphics Method:

Methods:

Return Type	Name	Description
void	drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, ImageObserver observer)	Draws an image based on source and destination points. img – the image to be drawn The source points define a rectangle of the pixels you want to draw: (sx1,sy1) – is the top left corner (sx2,sy2) – is the bottom right corner The destination points define where point 1 and point 2 will end up: (dx1,dy1) – is where source point one will end up (dx2,dy2) – is where source point one will end up

DrawImage Examples:

Source	Destination
	
	
	

Colors Overview:

Attributes:

Type	Name	Description
int	BLACK	Value for black
int	BLUE	Value for blue
int	CYAN	Value for cyan
int	DARK_GRAY	Value for dark gray
int	GRAY	Value for gray
int	GREEN	Value for green
int	LIGHT_GRAY	Value for light gray
int	MAGENTA	Value for magenta
int	ORAGNE	Value for orange
int	PINK	Value for pink
int	RED	Value for red
int	WHITE	Value for white
int	YELLOW	Value for yellow

Constructors:

Name	Description
Color(int r, int g, int b)	Creates a Color with the given (r,g,b) value r – (0-255) g – (0-255) b – (0-255) 0 color off 255 color fully on
Color(int r, int g, int b, int a)	Creates a Color with the given (r,g,b,a). a is how transparent the color is. a – (0-255) 0 fully transparent 1 fully opaque
Color(int rgb)	Creates a color from the given value
Color(int rgba, boolean hasalpha)	Creates a color from the given value. If hasalpha is true it will pull the transparency value from rgba

Methods:

Return Type	Name	Description
Color	brighter()	Creates a brighter version of the calling Color
Color	darker()	Creates a darker version of the calling color.
int	getAlpha()	Returns the alpha value of the color
int	getBlue()	Returns the blue value of the color
int	getRed()	Returns the red value of the color
int	getGreen()	Returns the green value of the color
int	getTransparency()	Returns the transparency mode.

Rotating Images:

Rotating an image is fairly complicated. We will not be covering how each line of code for rotating works.

The below code will rotate images for you:

```
// img will be the image being rotated
// angle will be the angle for the rotation

// corrects any angle bigger than 360
angle = angle%360;

// Creates the transform for the rotation
AffineTransform affineTransform = new AffineTransform();
affineTransform.setToTranslation(0,0);
affineTransform.rotate(Math.toRadians(angle), img.getWidth()/2, img.getHeight()/2);

// Stores the transparency of the original image
int transparency = img.getColorModel().getTransparency();

// Creates an image to store the rotated version of the original image
BufferedImage rotated =
        new BufferedImage( img.getWidth(), img.getHeight(), transparency);

// gets the graphics of the destination image.
Graphics2D  g = (Graphics2D)  (rotated.getGraphics());

// draws the original image onto the destination image, with the correct rotation
g.drawImage(img, affineTransform, null);
```