

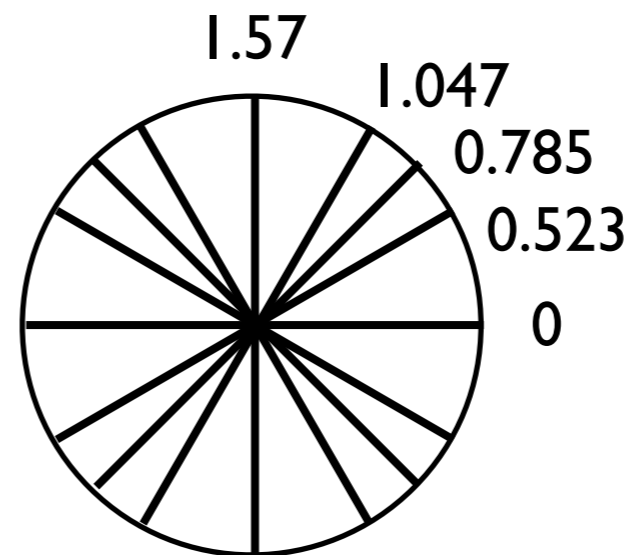
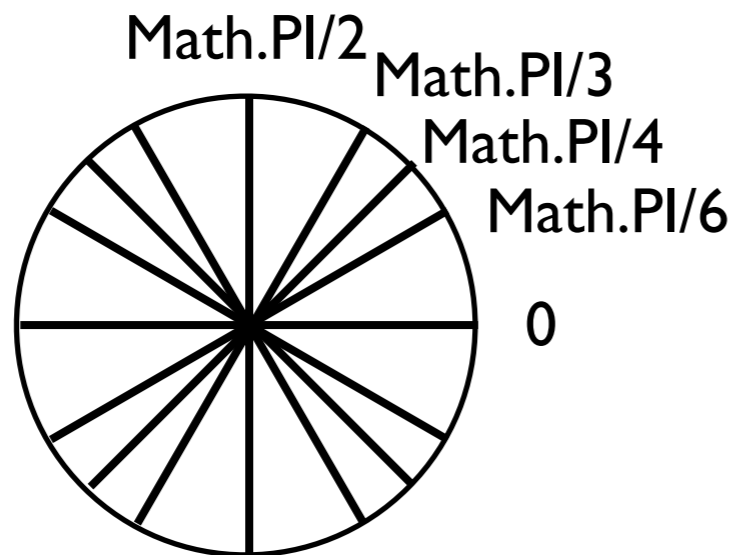
Basic Graphical Transformations

by Mr. F

Image Rotation

```
public void render(Graphics g)           (ulX, ulY)
{
    Graphics2D g2 = (Graphics2D)g;

    g2.rotate(rads, imageCenterX, imageCenterY);
    g2.drawImage(img, imageULX, imageULY, null);
    g2.rotate(-rads, imageCenterX, imageCenterY);
}
```



Drawing an Image with Transparency

```
public void render(Graphics g)
{
    Graphics2D g2 = (Graphics2D)g;
    Composite oldComp = g2.getComposite();
    AlphaComposite newComp;

    newComp = AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 0.25);
    g2.setComposite(newComp);
    g2.drawImage(image, 0, 0, null);

    g2.setComposite(oldComp);
}
```

This will draw the image with a 25% transparency.

ARGB

- Colors in Java are generally represented using the ARGB format
- 8 bits of information for the alpha (transparency), red, green and blue components
- That's a total of 256 different values (from 0 to 255)

ARGB and Hex

- ARGB values are generally represented in hexadecimal
- 8 bits is represented by 2 hexadecimal digits
- 2A is the same as $2*16 + 10 = 42$
- FF is the same as $15*16 + 15 = 255$
- 00 is the same as $0*16 + 0 = 0$

Examples

```
int onlyRed = 0xFFFF0000; //0x means this is hexadecimal
```

```
int onlyBlue = 0xFF0000FF;
```

```
int transparentGreen = 0x8000FF00;
```

<http://www.colorspire.com/rgb-color-wheel/>

Bitwise Commands

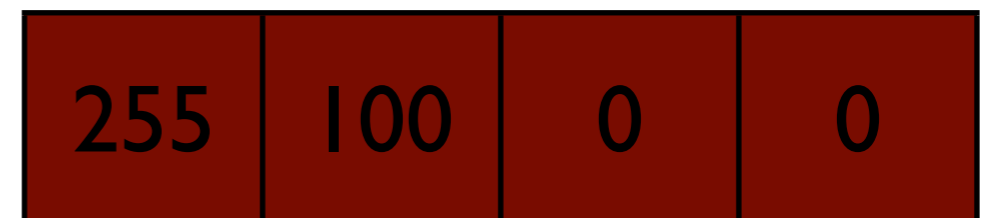
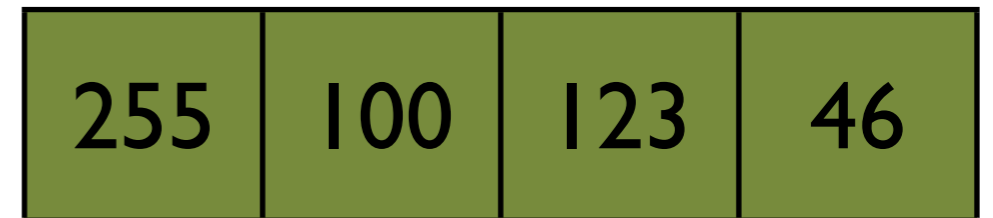
- Image editors makes use of the & and | bitwise commands
- The & operator is used for finding out what components of colors are **shared**, while the | operator is used for combining (but not mixing) the components of two colors

Bitwise Example

```
int argb = ...;  
int onlyRed = 0xFFFF0000;
```

```
argb = argb & onlyRed;
```

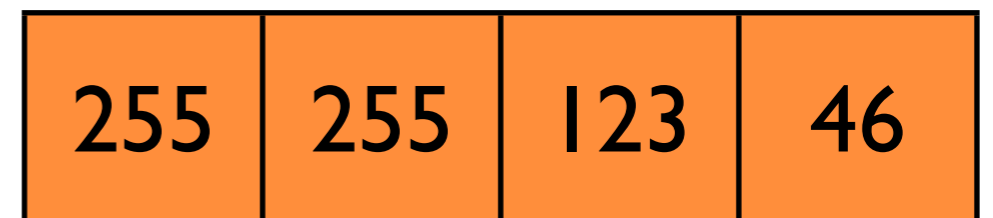
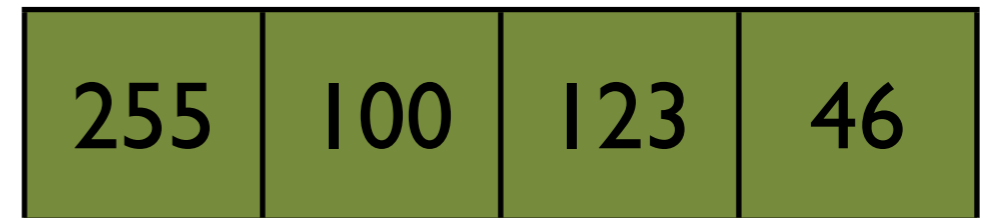
Only the overlapping red component of the color remains!



Bitwise Example

```
int argb = ...;  
int onlyRed = 0xFFFF0000;
```

```
argb = argb | onlyRed;
```



Adds a full red component to
the color

Bitwise Commands 2

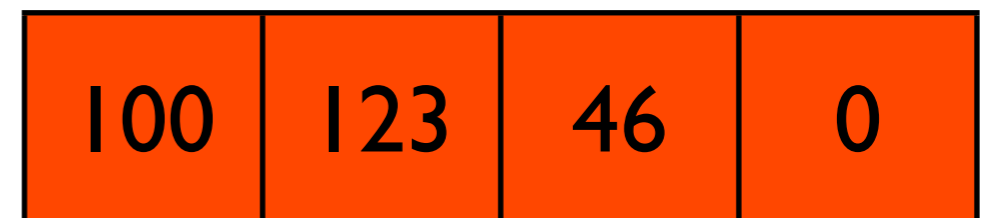
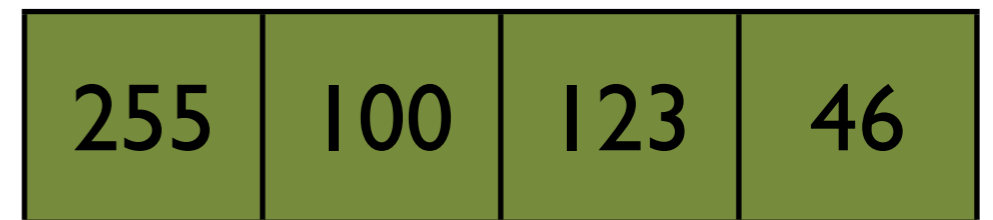
- The `>>` and `<<` operators can be used to shift a binary number a given number of bits to the right or left, respectively.

```
int x = 33;    //100001
x = x >> 2;    //001000
x = x << 2;    //100000
```

Another Bitwise Example

```
int argb = ...;  
arg = argb << 8;
```

A partially transparent color



Filtering an Image

- Create an ImageFilter object (several types)
- Run your image through the filter, which produces a new image that has been modified

Example

RGBImageFilter is abstract

```
ImageFilter filter = new RGBImageFilter()
{
    public int filterRGB(int x, int y, int rgb) {
        return rgb & 0xFFFF0000; //only red
    }
};
```

```
ImageProducer prod = new FilteredImageSource(img.getSource(), filter);
Image filteredImage = Toolkit.getDefaultToolkit().createImage(prod);
g.drawImage(filteredImage, 0, 0, null);
```

Redify (Part I)

```
public class MyBadGuy
{
    private int filter = 0xFFFFFFFF; //everything is in
    private boolean redify = true;
    private ImageFilter imgFilter;
    private BufferedImage image;

    public void update() {
        if(redify) {
            filter -= 0x00000101; //subtracts one off from green and blue
            if(filter % 256 == 0)
                redify = false;
        }
        else {
            filter += 0x00000101; //subtracts one off from green and blue
            if(filter % 256 == 255)
                redify = true;
        }
    }
}
```

Redify (Part II)

```
public class MyBadGuy
{
    public MyBadGuy(...)
    {
        super(...);
        image = ...;
        imgFilter = new RGBImageFilter()
        {
            public int filterRGB(int x, int y, int rgb) {
                return rgb & filter;
            }
        };
    }

    public void render(Graphics g)
    {
        ImageProducer prod = new FilteredImageSource(image.getSource(), imgFilter);
        Image filteredImage = Toolkit.getDefaultToolkit().createImage(prod);
        g.drawImage(filteredImage, 0, 0, null);
    }
}
```