

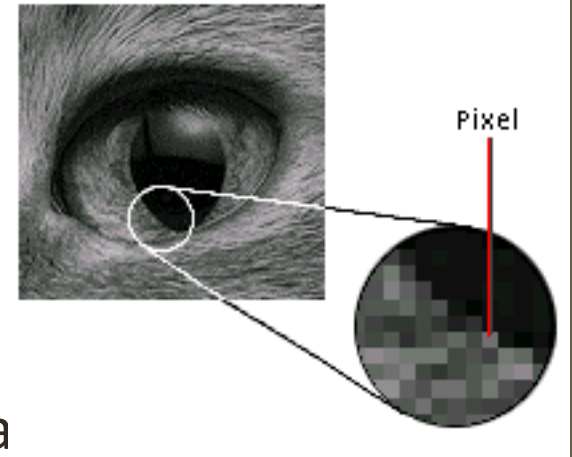
# Prática II: Processamento de Imagens em Java

INE5431 Sistemas Multimídia

# Objetivos da Aula Prática

- Reforçar conceitos básicos em representação digital de imagens
- Entender mais sobre o processamento de imagens digitais usando Java
- Suposições
  - Conhecimento de Java: Swing

# Imagens Digitais

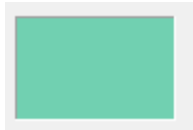


- Formatos de Imagens
  - Imagens no computador são representadas por bitmaps
    - bitmap = matriz espacial bidimensional de elementos de imagem chamados de pixels
      - reticulado - cada elemento da matriz possui uma informação referente à cor associada aquele ponto específico
    - pixel é o menor elemento de resolução da imagem
      - tem um valor numérico chamado amplitude
      - define ponto preto e branco, nível de cinza, ou atributo de cor (3 valores)
      - Expresso por um número de bits
        - 1 para imagens P&B, 2, 4, 8, 12, 16 ou 24 bits
- “Resolução” da imagem é o número de elementos que a imagem possui na horizontal e na vertical

# Imagens Digitais


- Convertendo imagens RGB em tons de cinza

- $Y = 0.3R + 0.59G + 0.11B;$



Vermelho:	113
Verde:	208
Azul:	177

$$Y = 0,3*113+0,59*208+0,11*177=176$$



- Convertendo imagens tons de cinza em binárias

- Pixel é preto se tom de cinza é abaixo da metade da escala e branco se o tom de cinza é acima da metade da escala

- Exemplo: imagens de 256 tons:

- $y \geq 127$  é branco
    - $Y < 127$  é preto

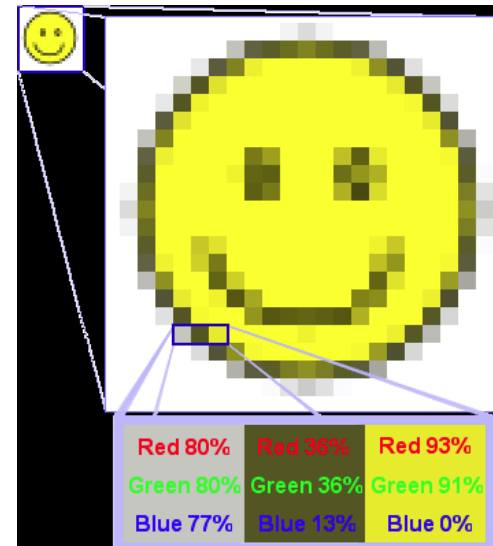
# Classes Java

- BufferedImage
  - Imagem padrão do awt
  - `public BufferedImage(int width, int height, int imageType)`  
Constrói um BufferedImage de um dos tipos de imagens predefinidas.
  - Parametros:
    - width – largura da imagem criada
    - height – altura da imagem criada
    - imageType – tipo da imagem criada
  - Alguns tipos de imagens:
    - `TYPE_BYTE_BINARY`: Imagens de 1, 2, ou 4 bits.
    - `TYPE_BYTE_GRAY`: Tons de cinza, 1 octeto por pixel
    - `TYPE_INT_RGB`: imagens RGB de 8 bits por componente

# Classes Java

- BufferedImage
  - Leitura de um arquivo de imagem:
    - `BufferedImage bimg= ImageIO.read(URL url)`
    - `BufferedImage bimg= ImageIO.read(new File(<nome>))`
  - Lendo pixels em imagens `BufferedImage.TYPE_INT_RGB`:
    - `int rgb = bufferedImage.getRGB(w,h);`
    - `int r = (int)((rgb&0x00FF0000)>>>16);`  
// componente vermelho
    - `int g = (int)((rgb&0x0000FF00)>>>8);`  
// componente verde
    - `int b = (int)(rgb&0x000000FF);`  
//componente azul

Sample Length:	8								8								8								8							
Channel Membership:	Alpha								Red								Green								Blue							
Bit Number:	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

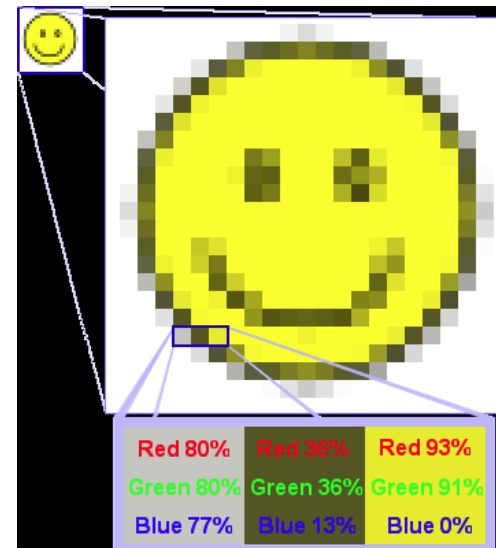


# Classes Java

- BufferedImage
  - Manipulando pixels em imagens BufferedImage.TYPE\_INT\_RGB:

- WritableRaster raster = img.getRaster();
    - Raster é o bitmap (mapa de bits)

```
for(int h=0;h<100;h++)  
    for(int w=0;w<100;w++) {  
        // Componente Vermelho  
        raster.setSample(w,h,0,220);  
        // Componente Verde  
        raster.setSample(w,h,1,219);  
        // Componente Azul  
        raster.setSample(w,h,2,97);} 
```



# Classes Java

- BufferedImage
  - Manipulando pixels em imagens BufferedImage.TYPE\_INT\_RGB:

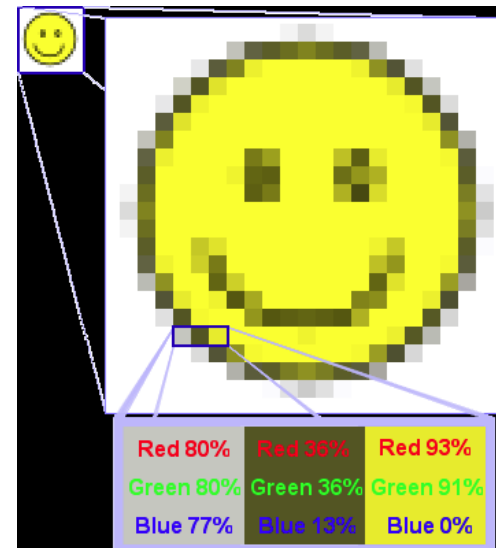
```
BufferedImage img = ...
```

```
int rgb = 0xFFDCDB61; // Opção de construir uma imagem verde
```

```
for(int h=0;h<100;h++)
```

```
    for(int w=0;w<100;w++)
```

```
        img.setRGB(h,w,rgb);
```





# Classes Java

- BufferedImage
  - Manipulando pixels em imagens TYPE\_BYTE\_GRAY:
    - Apenas o `raster.setSample(w,h,0,0xFF);` // branco
  - Manipulando pixels em imagens binárias TYPE\_BYTE\_BINARY:
    - Apenas o `raster.setSample(w,h,0,0);` // preto
    - Apenas o `raster.setSample(w,h,0,1);` // branco

# Apresentação do código oferecido

```
public class JImagePanel extends JPanel{  
    private BufferedImage image;  
    int x, y;  
    public JImagePanel(BufferedImage image, int x, int y) {  
        super();  
        this.image = image;  
        this.x = x;  
        this.y = y;  
    }  
    protected void paintComponent(Graphics g) {  
        super.paintComponent(g);  
        g.drawImage(image, x, y, null);  
    }  
}
```

# Apresentação do código oferecido

```
public class ImageApp {  
    public static void main(String[] args) {  
        ImageApp ia = new ImageApp();  
        BufferedImage imgJPEG =  
            loadImage("http://www.inf.ufsc.br/~willrich/smil/midias/...  
                                imagens/elephant.jpg");  
  
        BufferedImage imgRGB = criaImagemRGB();  
        BufferedImage imgCinza = criaImagemCinza();  
        BufferedImage imgBinaria = criaImagemBinaria();  
        ia.apresentaImagem(new JFrame("imgJPEG"), imgJPEG);  
        ia.apresentaImagem(new JFrame("imgRGB"), imgRGB);  
        ia.apresentaImagem(new JFrame("imgCinza"), imgCinza);  
        ia.apresentaImagem(new JFrame("imgBinaria"), imgBinaria);  
        imprimePixeis(imgJPEG);  
    }  
}
```

# Apresentação do código oferecido

```
public class ImageApp {  
    ...  
    public static BufferedImage loadImage(String surl) {  
        BufferedImage bimg = null;  
        try {  
            URL url = new URL(surl);  
            bimg = ImageIO.read(url);  
            //bimg = ImageIO.read(new File("D:/Temp/mundo.jpg"));  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
        return bimg;  
    }  
}
```

# Apresentação do código oferecido

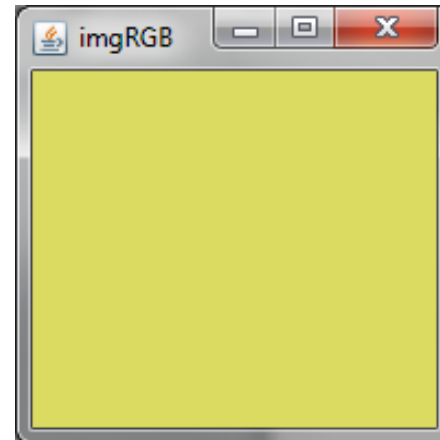
```
public class ImageApp {
```

```
...
```

```
    public apresentImagem(JFrame frame, BufferedImage img) {  
        frame.setBounds(0, 0, img.getWidth(), img.getHeight());  
        JPanel panel = new JPanel(img, 0, 0);  
        frame.add(panel);  
        frame.setVisible(true);  
    }
```

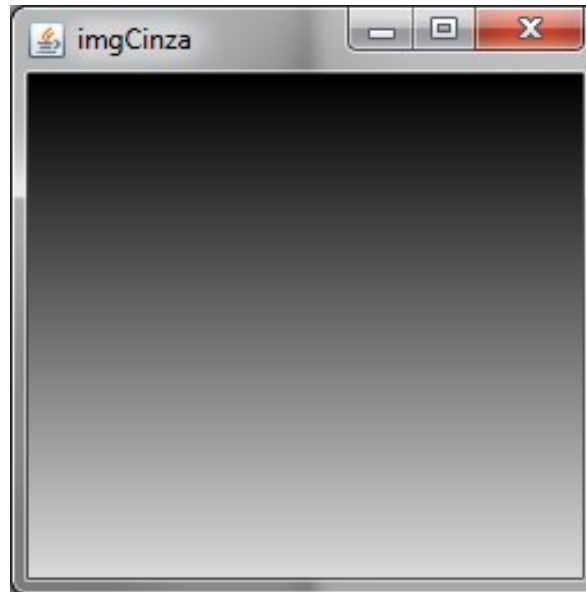
# Apresentação do código oferecido

```
public static BufferedImage criaImagemRGB() {  
    BufferedImage img = new BufferedImage(200, 200,  
        BufferedImage.TYPE_INT_RGB);  
    WritableRaster raster = img.getRaster();  
    for(int h=0;h<img.getHeight();h++)  
        for(int w=0;w<img.getWidth();w++) {  
            raster.setSample(w,h,0,220); // Componente Vermelho  
            raster.setSample(w,h,1,219); // Componente Verde  
            raster.setSample(w,h,2,97);  // Componente Azul  
        }  
    return img;  
}
```



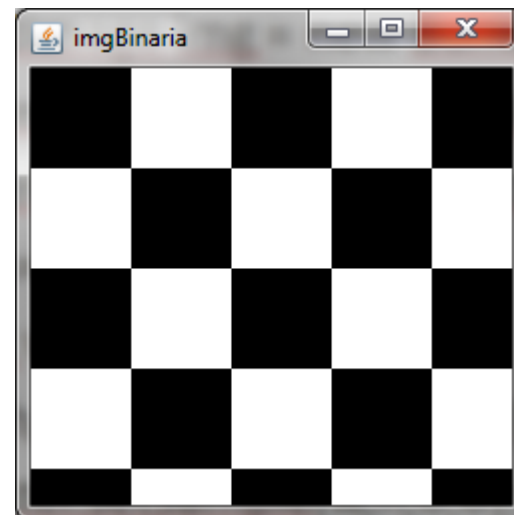
# Apresentação do código oferecido

```
public static BufferedImage criaImagemCinza() {  
    BufferedImage img = new BufferedImage(256, 256,  
        BufferedImage.TYPE_BYTE_GRAY);  
    WritableRaster raster = img.getRaster();  
    for(int h=0;h<img.getHeight();h++)  
        for(int w=0;w<img.getWidth();w++) {  
            raster.setSample(w,h,0,h);  
        }  
    return img;  
}
```



# Apresentação do código oferecido

```
public static BufferedImage criaImagemBinaria() {  
    // Cria imagem de 256x256 de uma cor sólida  
    BufferedImage img = new BufferedImage(256, 256,  
        BufferedImage.TYPE_BYTE_BINARY);  
    WritableRaster raster = img.getRaster();  
    for(int h=0;h<img.getHeight();h++)  
        for(int w=0;w<img.getWidth();w++) {  
            if (((h/50)+(w/50)) % 2 == 0)  
                raster.setSample(w,h,0,0); // checkerboard pattern.  
            else raster.setSample(w,h,0,1);  
        }  
    return img;  
}
```





# Apresentação do código oferecido

```
// Imprime valores dos pixels de imagem RGB
public static void imprimePixeis(BufferedImage bufferedImage) {
    // Obtém resolução da imagem
    int width = bufferedImage.getWidth();
    int height = bufferedImage.getHeight();

    for(int h=0;h<height;h++)
        for(int w=0;w<width;w++) {
            int rgb = bufferedImage.getRGB(w,h);
            int r = (int)((rgb&0x00FF0000)>>>16); // componente vermelho
            int g = (int)((rgb&0x0000FF00)>>>8); // componente verde
            int b = (int)(rgb&0x000000FF); // componente azul
            System.out.print("at (" +w+", "+h+"): ");
            System.out.println(r+", "+g+", "+b);
        }
    }
```