CS 10:
Problem solving via Object Oriented Programming

Graphics

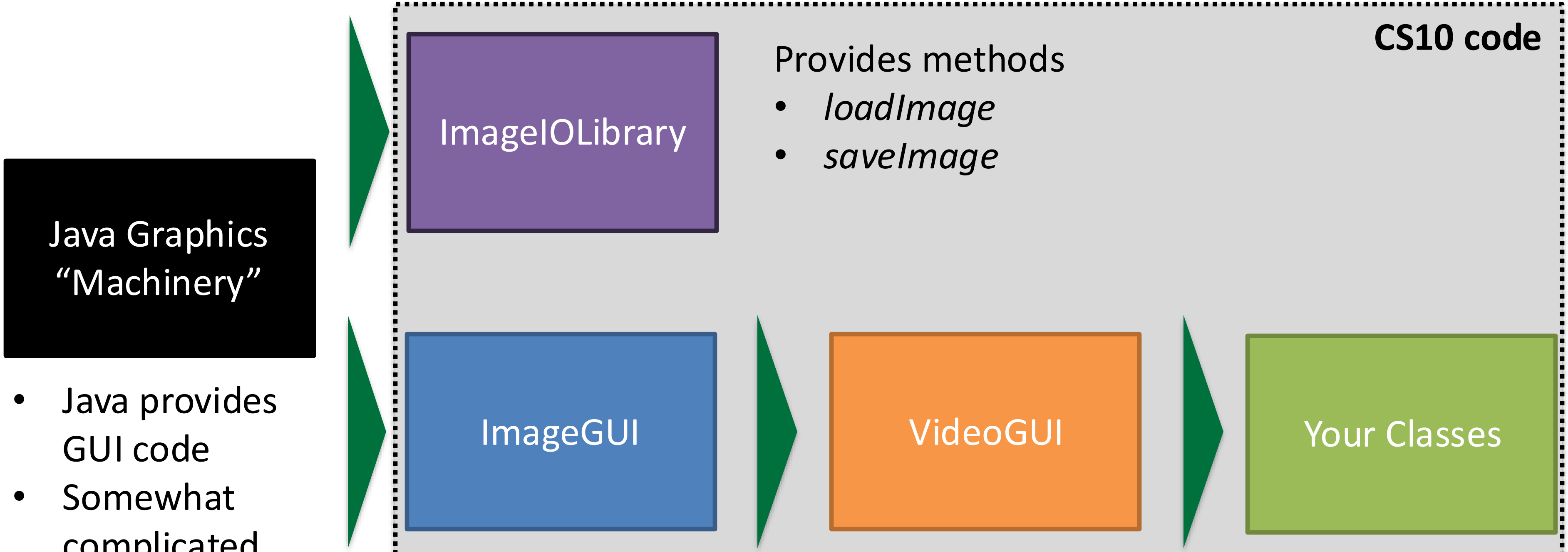
Agenda

1. Images
2. Video
3. Sample applications

Key points:

1. **Images are made up of pixels**
2. **Each pixel is a Color object**
3. **Color objects can manipulate red, green, and blue components**

I’ve provided some code to handle the messy parts of Java’s graphics “machinery”



- Java provides GUI code
- Somewhat complicated
- Learning the specifics of Java’s GUI “machinery” not really the point of this course
- Provides
  - *BufferedImage*
  - *JFrame*

Provides methods
- *loadImage*
- *saveImage*

Display one image on screen or two images side by side
- *setImage1*
- *setImage2*

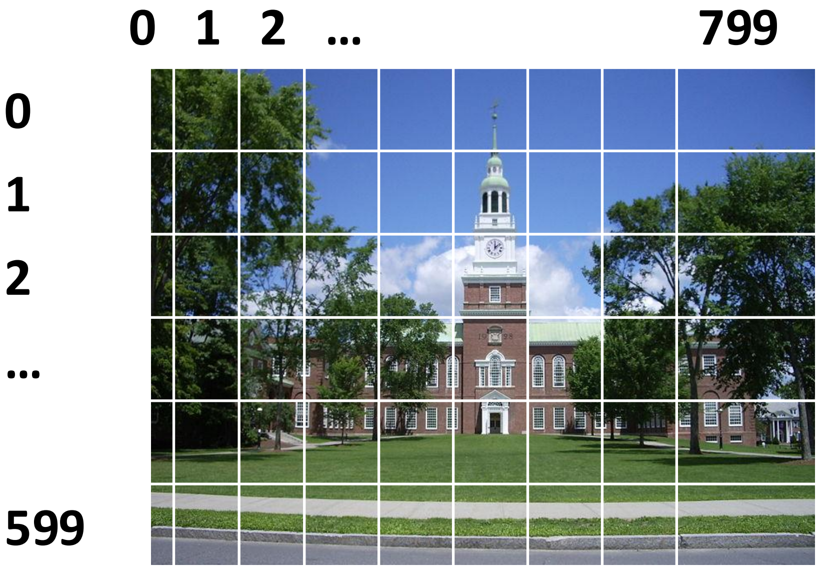
- Inherits from *ImageGUI*
- Sets up camera to take snapshot every 100ms
- Displays camera image using *ImageGUI* *setImage1* method

You inherit from *VideoGUI,* get video feed (and more)

Java provides the *BufferedImage* class to hold images in memory

800 x 600 image

NOTE Y axis counts downward!

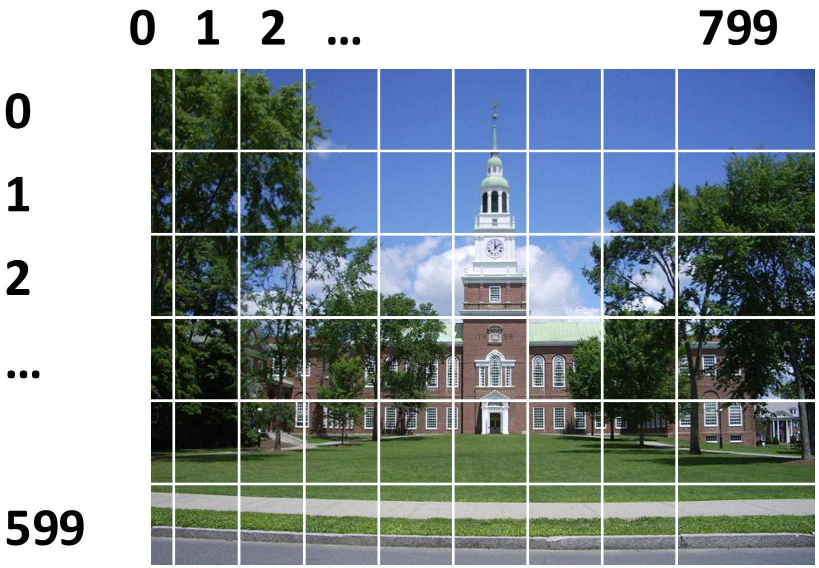


- I've provided a simple *ImageIOLibrary* class to load and save *BufferedImages*
    - Use *ImageIOLibrary.loadImage* to read images from disk into a *BufferedImage*
    - Use *ImageIOLibrary.saveImage* to write a *BufferedImage* to disk
- *BufferedImages* are comprised of pixels at x,y locations on the image
- Pixels are represented by Java-provided *Color* objects
- *Color* objects tell Java what color to render at position x,y

Images are made up of pixels, each with a (x,y) location and a color

800 x 600 image

NOTE Y axis counts downward!



Load image from disk into a *BufferedImage img*

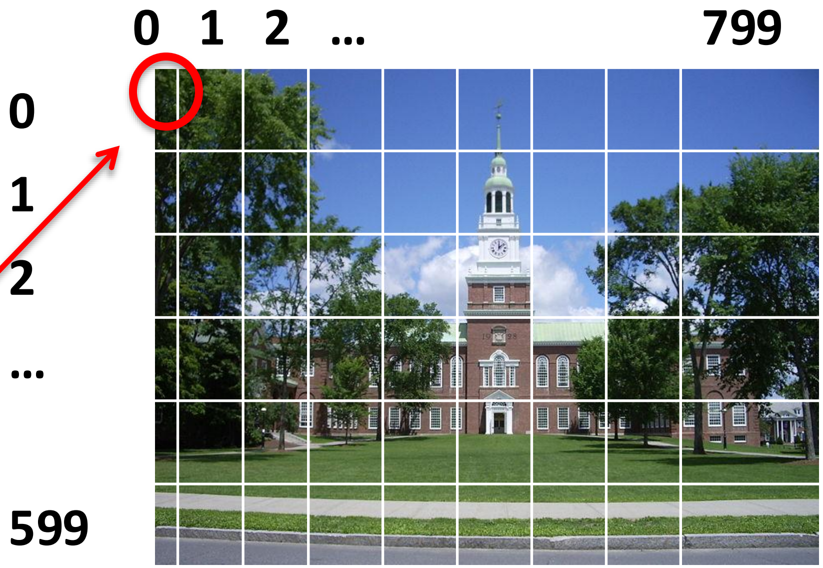
Note: working directory is the project directory!

```
int x = 0, y = 0;
BufferedImage img = ImageIOLibrary.loadImage("pictures/baker.png");
Color colorBelow = new Color(img.getRGB(x,y+1));
img.setRGB(x,y,colorBelow.getRGB());
```

Images are made up of pixels, each with a (x,y) location and a color

800 x 600 image

NOTE Y axis counts downward!



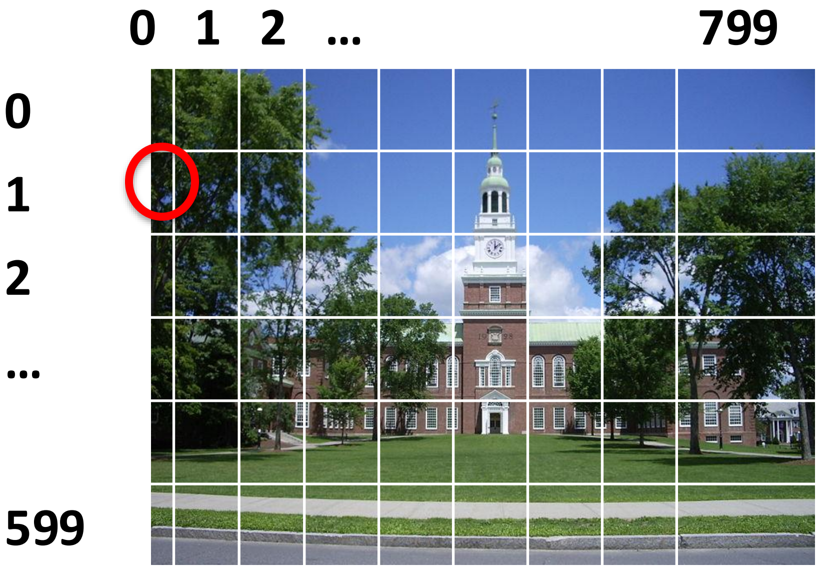
```
int x = 0, y = 0;
BufferedImage img = ImageIOLibrary.loadImage("pictures/baker.png");
Color colorBelow = new Color(img.getRGB(x,y+1));
img.setRGB(x,y,colorBelow.getRGB());
```

Get color at location x,y+1 using *getRGB* method of *BufferedImage* object

Images are made up of pixels, each with a (x,y) location and a color

800 x 600 image

NOTE Y axis counts downward!



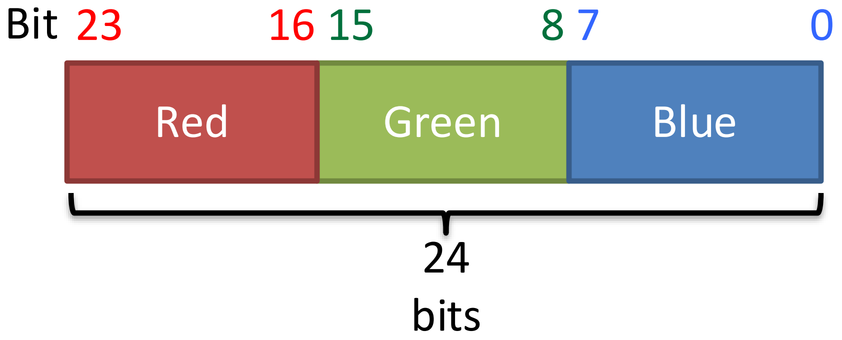
```
int x = 0, y = 0;
BufferedImage img = ImageIOLibrary.loadImage("pictures/baker.png");
Color colorBelow = new Color(img.getRGB(x,y+1));
img.setRGB(x,y,colorBelow.getRGB());
```

Set color at x,y to new color using *setRGB*

See DrawSquare.java for demo on how to draw a square on the screen

See FadeIn.java for copying colors from one image to another

Behind the scenes, Java represents colors as a 24-bit integer



Java uses a 24-bit integer to represent red, green, and blue color component intensity

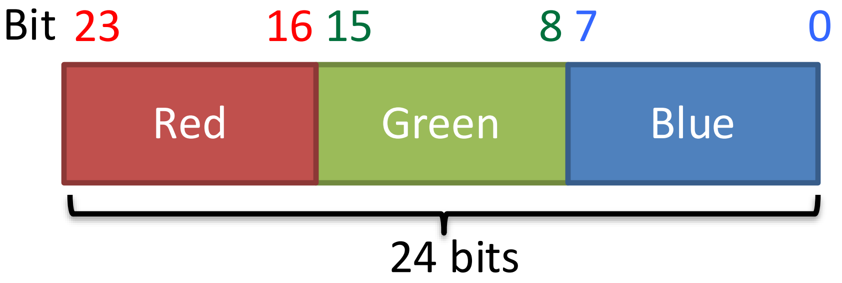
Each color component has 8 bits, so intensity range for each component is 0-255:

0 = no color

255 = max color

Java provides a convenient Color class to store color values

Pixel colors are made up of Red, Green, and Blue components of varying intensity



Each R,G, or B components has 8 bits to control color intensity

8 bits means intensity range 0-255

Red	Green	Blue	Result
255	255	255	**White**
0	0	0	**Black**
255	0	0	**Bright red**
0	255	0	**Bright green**
0	0	255	**Bright blue**
128	0	0	**Not-as-bright-red**
0	128	0	**Not-as-bright green**
0	0	128	**Not-as-bright-blue**

All colors full on

All colors off

One color full on, others off

One color half on, others off

- **Human eye is unlikely to notice a very small change in color**
- **Useful for SA-3**

Java’s Color class makes it easy to manipulate pixel colors

ImageDimmer.java

```
public class ImageDimmer {

  public BufferedImage dimImage(BufferedImage originalImage) {
    //create blank image of the same size as the original
    BufferedImage dimmedImage = new BufferedImage(originalImage.getWidth(), originalImage.getHeight(), BufferedImage.TYPE_INT_ARGB);

    //dim each pixel
    for (int y = 0; y < originalImage.getHeight(); y++) {
      for (int x = 0; x < originalImage.getWidth(); x++) {
        // Get current color; scale each channel (but don't exceed 255); put new color
        Color color = new Color(originalImage.getRGB(x, y));
        int red = color.getRed()/2;
        int green = color.getGreen()/2;
        int blue = color.getBlue()/2;
        Color newColor = new Color(red, green, blue);
        dimmedImage.setRGB(x, y, newColor.getRGB());
      }
    }
    return dimmedImage;
  }

  public static void main(String[] args) {
    //load image and dim each pixel
    BufferedImage originalImage = ImageIOLibrary.loadImage("pictures/baker.png");
    ImageDimmer dimmer = new ImageDimmer();
    BufferedImage dimmedImage = dimmer.dimImage(originalImage);

    //display results side by side
    ImageGUI gui = new ImageGUI("Dimmed", originalImage, dimmedImage);

  }
```

Load *BufferedImage* from image on disk using ImageIOLibrary

Dim each pixel on the loaded image

Java's Color class makes it easy to manipulate pixel colors

ImageDimmer.java

```java
public class ImageDimmer {

  public BufferedImage dimImage(BufferedImage originalImage) {
    //create blank image of the same size as the original
    BufferedImage dimmedImage = new BufferedImage(originalImage.getWidth(), originalImage.getHeight(), BufferedImage.TYPE_INT_ARGB);

    //dim each pixel
    for (int y = 0; y < originalImage.getHeight(); y++) {
      for (int x = 0; x < originalImage.getWidth(); x++) {
        // Get current color; scale each channel (but don't exceed 255); put new color
        Color color = new Color(originalImage.getRGB(x, y));
        int red = color.getRed()/2;
        int green = color.getGreen()/2;
        int blue = color.getBlue()/2;
        Color newColor = new Color(red, green, blue);
        dimmedImage.setRGB(x, y, newColor.getRGB());
      }
    }
    return dimmedImage;
  }

  public static void main(String[] args) {
    //load image and dim each pixel
    BufferedImage originalImage = ImageIOLibrary.loadImage("pictures/baker.png");
    ImageDimmer dimmer = new ImageDimmer();
    BufferedImage dimmedImage = dimmer.dimImage(originalImage);

    //display results side by side
    ImageGUI gui = new ImageGUI("Dimmed", originalImage, dimmedImage);

}
```

Create a blank image of the same size as the original so we don't alter the original image, use *getWidth* and *getHeight*

Loop over every pixel (nested loop)

Get color at each x,y location in original
Dim by dividing red, green, blue components by 2
Decimal component after division? Dropped! Cast double to integer

Set location x,y on image copy to dimmed color

Returned dimmed image

Java’s Color class makes it easy to manipulate pixel colors

ImageDimmer.java

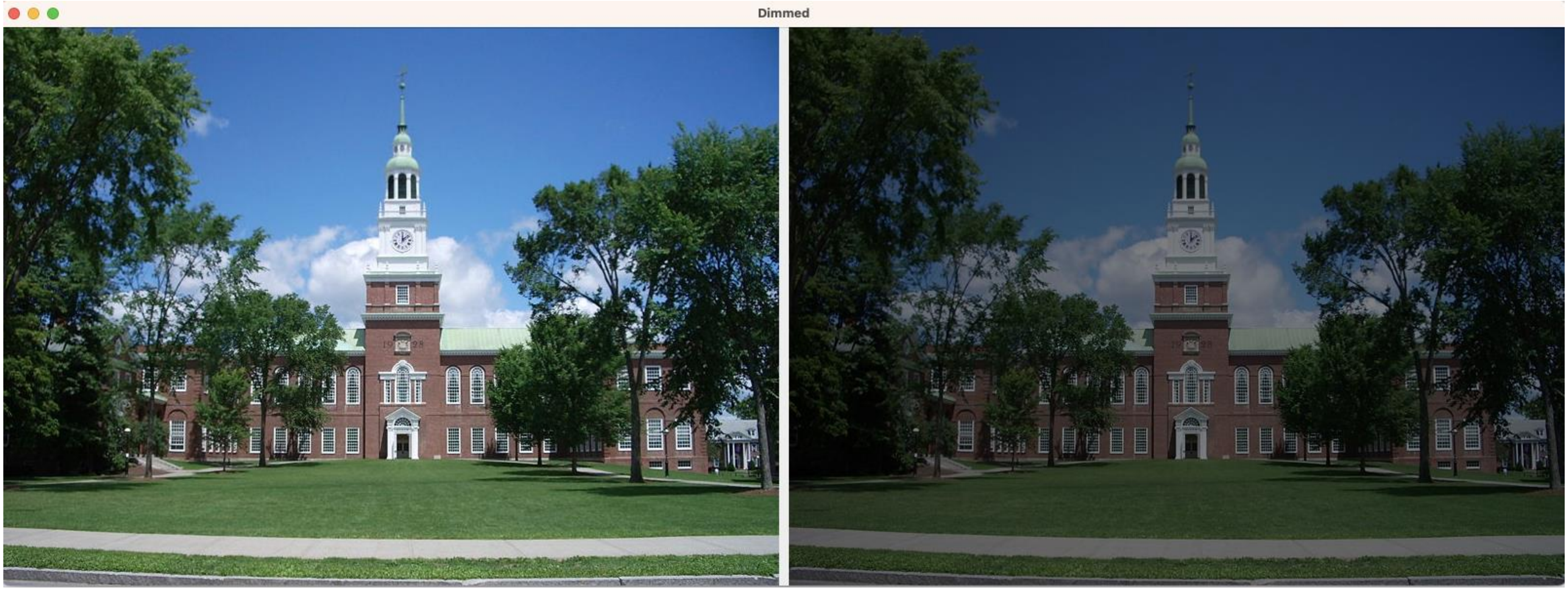
```
public class ImageDimmer {

   public BufferedImage dimImage(BufferedImage originalImage) {
      //create blank image of the same size as the original
      BufferedImage dimmedImage = new BufferedImage(originalImage.getWidth(), originalImage.getHeight(), BufferedImage.TYPE_INT_ARGB);

      //dim each pixel
      for (int y = 0; y < originalImage.getHeight(); y++) {
         for (int x = 0; x < originalImage.getWidth(); x++) {
            // Get current color; scale each channel (but don't exceed 255); put new color
            Color color = new Color(originalImage.getRGB(x, y));
            int red = color.getRed()/2;
            int green = color.getGreen()/2;
            int blue = color.getBlue()/2;
            Color newColor = new Color(red, green, blue);
            dimmedImage.setRGB(x, y, newColor.getRGB());
         }
      }
      return dimmedImage;
   }

   public static void main(String[] args) {
      //load image and dim each pixel
      BufferedImage originalImage = ImageIOLibrary.loadImage("pictures/baker.png");
      ImageDimmer dimmer = new ImageDimmer();
      BufferedImage dimmedImage = dimmer.dimImage(originalImage);

      //display results side by side
      ImageGUI gui = new ImageGUI("Dimmed", originalImage, dimmedImage);

   }
```

Display images side by side

- ***ImageGUI* can display one or two images (CS10 code, not provided by Java)**
- **Provide one *BufferedImage* in constructor to see one image**
- **Provide two *BufferedImages* to see both side by side**

Java’s Color class makes it easy to manipulate pixel colors

```
public class ImageDimmer {
```



```
  public static void main(String[] args) {
    //load image and dim each pixel
    BufferedImage originalImage = ImageIOLibrary.loadImage("pictures/baker.png");
    ImageDimmer dimmer = new ImageDimmer();
    BufferedImage dimmedImage = dimmer.dimImage(originalImage);

    //display results side by side
    ImageGUI gui = new ImageGUI("Dimmed", originalImage, dimmedImage);

  }
```

Display both original and dimmed images with *ImageGUI*

BlurImage averages around each pixel in the image using two nested loops

BlurImage.java

```java
public static void main(String[] args) {
    int radius = 1; //average r row above to r rows below, r cols left to r cols right

    //load image and create a blank image called result
    BufferedImage image = ImageIOLibrary.loadImage("pictures/baker.png");
    BufferedImage result = new BufferedImage(image.getWidth(), image.getHeight(), BufferedImage.TYPE_INT_ARGB);

    // Nested loop over every pixel in original image
    for (int y = 0; y < image.getHeight(); y++) {
        for (int x = 0; x < image.getWidth(); x++) {
            int sumR = 0, sumG = 0, sumB = 0;
            int n = 0;
            // Nested loop over neighbors
            // but be careful not to go outside image (max, min stuff).
            for (int ny = Math.max(0, y - radius);  ny < Math.min(image.getHeight(), y + 1 + radius);  ny++) {
                for (int nx = Math.max(0, x - radius);  nx < Math.min(image.getWidth(), x + 1 + radius);  nx++) {
                    // Add all the neighbors (& self) to the running totals
                    Color c = new Color(image.getRGB(nx, ny));
                    sumR += c.getRed();
                    sumG += c.getGreen();
                    sumB += c.getBlue();
                    n++;
                }
            }
            Color newColor = new Color(sumR / n, sumG / n, sumB / n);
            result.setRGB(x, y, newColor.getRGB());
        }
    }

    //display images
    ImageGUI gui = new ImageGUI("Blurred image", image, result);
}
```

Load image and make blank called *result*

Loop over each pixel in *image* using a nested loop

Sum red, green, and blue components for this pixel's neighbors, also count neighbors

Double nested loops useful for PS-1

Loop *radius* rows above to *radius* rows below, and *radius* rows left to *radius* rows right using second nested loop

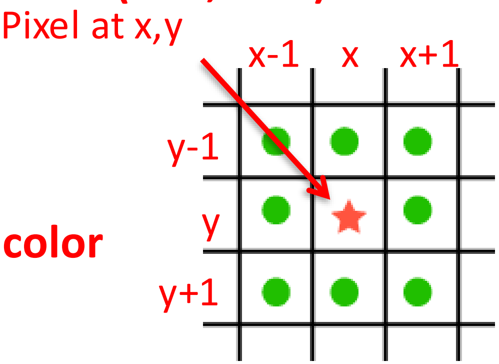
Don't go off screen (min, max)

Sum color components, increment neighbor count

Calculate average color

Fill result with averaged color

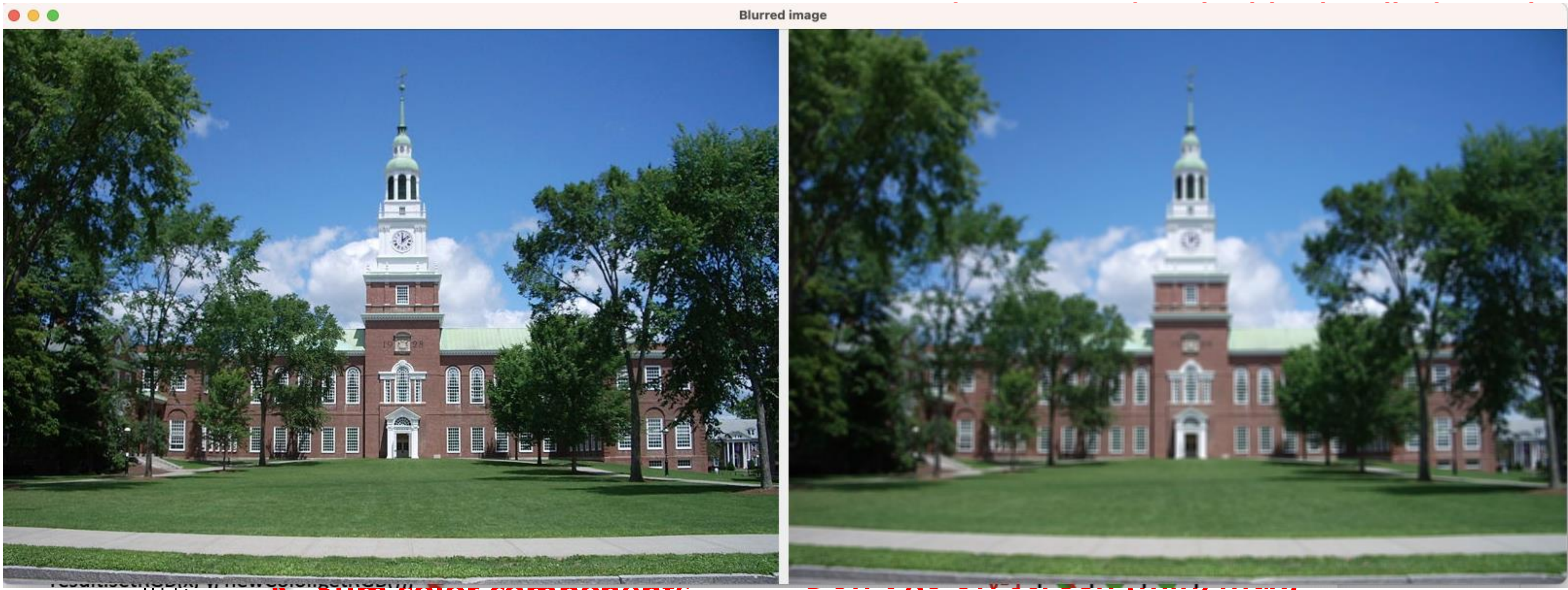
Display original and result



BlurImage averages around each pixel in the image using two nested loops

BlurImage.java

```
public static void main(String[] args) {
   int radius = 1; //average r row above to r rows below, r cols left to r cols right
```

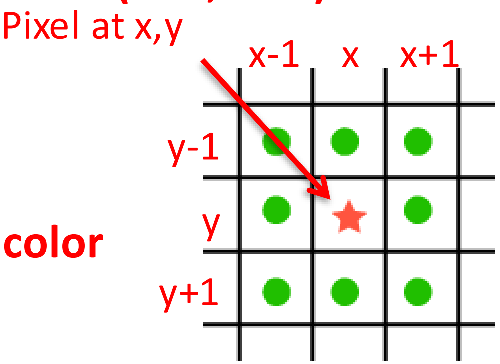


```
            }
         }
         Color newColor = new Color(sumR / n, sumG / n, sumB / n);
         result.setRGB(x, y, newColor.getRGB());
      }
   }

   //display images
   ImageGUI gui = new ImageGUI("Blurred image", image, result);
}
```

Sum color components, increment neighbor count

Calculate average color
Fill result with averaged color

Display original and result



Agenda

1. Images

2. Video

Key points:

1. **Video can be thought of as a sequence of images**
2. **Each image can be altered (just be done before next image arrives)**

3. Sample applications

Previously we manipulated a single image, video is just a stream of images over time

***n* images form a video**

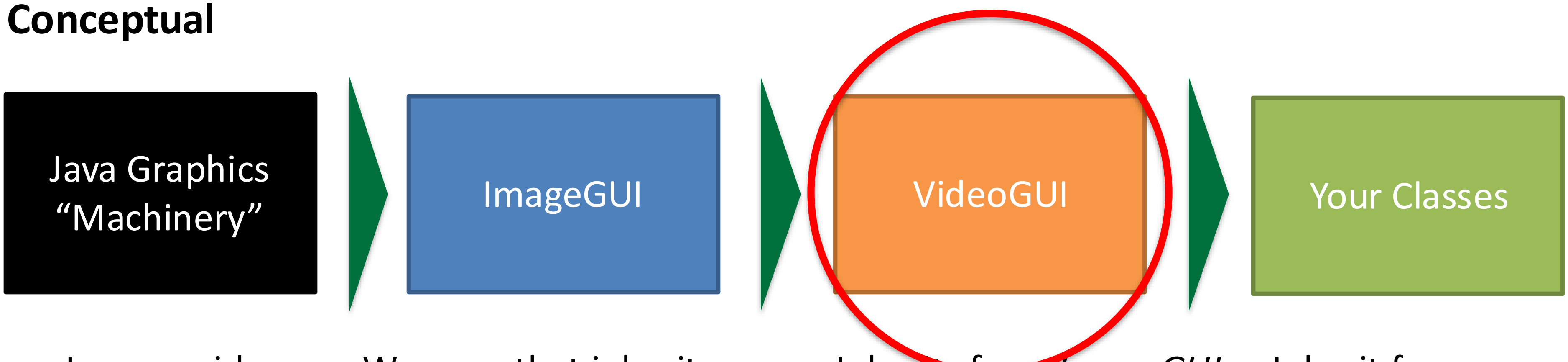
0 1 … n-2 n-1

- Can individually process each image (called a frame in video)
- Just need to be done processing before the next image arrives!
- Can do some tricks if we realize most of the image is the same from frame to frame

Image: https://www.elmedia-video-player.com/frame-by-frame-video-player-mac.html

I’ve provided a VideoGUI class to try to make handling video easier

Conceptual



Java Graphics “Machinery”

ImageGUI

VideoGUI

Your Classes

- Java provides GUI code
- Somewhat complicated
- Learning the specifics of Java’s GUI “machinery” not really the point of this course

- Wrapper that inherits from JFrame “machinery”
- Constructor takes one or two images
  - If one image display that image
  - If two images display both images side by side
- Update displayed images
  - *setImage1*
  - *setImage2*

- Inherits from *ImageGUI*
- Sets up camera to take snapshot every 100ms
- Provides methods we override:
  - *handleImage()*
  - *handleMousePress()*
  - *handleKeyPress()*
- By default, displays new camera image by calling *setImage1* and passing latest camera shot

- Inherit from *VideoGUI*
- Override *handleImage()* to handle frames as captured
- Can also override *handleMousePress* and *handleKeyPress*
- Get *ImageGUI*’s methods too!

Last image from camera is stored in instance variable *image*

VideoGUI.java

Inherit from ImageGUI which inherits from Java's graphics machinery

Camera set up different for Macs vs Windows

Downsize sample (for faster processing)
Here we make image half size

```
public class VideoGUI extends ImageGUI {
    protected boolean mac = true;                   // is this computer a mac?
    private static final double scale = 0.5;        // to downsize the image (for speed)
    private static final boolean mirror = true;     // mirror so image "looks right"

    protected BufferedImage image;                  // image grabbed from webcam (if any)
```

- **Last camera image stored here**
- **Updated every 100 ms as new images captured**

Mirror swaps left and right, makes things "look right"

handleImage allows image processing; also available *handleMousePress* and *KeyPress*

VideoGUI.java

Inherit from VideoGUI and override these methods for you own code

```
/**
 * Draws image instance variable filled by camera as left image on ImageGUI
 */
public void handleImage() {
   setImage1(image);
}
```

- ***handleImage*** **called by** ***VideoGUI*** **each time a new frame arrives**
- **By default it makes no changes to** ***image***
- **Sets** ***image1*** **on** ***ImageGUI*****, which updates window with new image**
- **We can override it to apply our changes**

```
/**
 * Called back when the mouse is pressed.
 */
public void handleMousePress(int x, int y) {
   System.out.println("Got mouse " + x + ", " + y);
}
```

- ***handleMousePress*** **called by** ***VideoGUI*** **when the mouse is pressed**
- **Returns mouse's x and y location on screen when pressed**

```
/**
 * Called back when a key is pressed
 */
public void handleKeyPress(char key) {
   System.out.println("Key pressed: " + key);
}
```

- ***handleKeyPress*** **called by** ***VideoGUI*** **when the key is pressed**
- **Returns the key that was pressed**

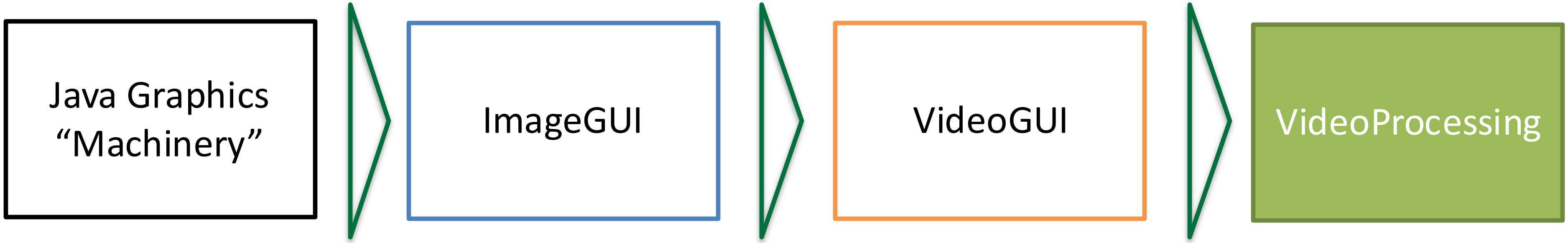
Agenda

1. Images

2. Video

3. Sample applications

Demo: VideoProcessing



Java Graphics “Machinery”

ImageGUI

VideoGUI

VideoProcessing

Notes:

- Alters each image taken by camera
- Acts after camera takes image and before image is displayed by overriding *handleImage*
- Brightens blue color component, dims red and green

VideoProcessing alters each image taken by the camera before it is displayed

VideoProcessing.java

```java
public class VideoProcessing extends VideoGUI {
    public VideoProcessing() {
        super("VideoProcessing");
     }

    public void scaleColor(double scaleR, double scaleG, double scaleB) {
        //safety check
        if (image == null || scaleR < 0 || scaleG < 0 || scaleB < 0) { return; }

        // Nested loop over every pixel
        for (int y = 0; y < image.getHeight(); y++) {
            for (int x = 0; x < image.getWidth(); x++) {
                // Get current color; scale each channel (but don't exceed 255); put new color
                Color color = new Color(image.getRGB(x, y));
                int red = (int)(Math.min(255, color.getRed()*scaleR));
                int green = (int)(Math.min(255, color.getGreen()*scaleG));
                int blue = (int)(Math.min(255, color.getBlue()*scaleB));
                Color newColor = new Color(red, green, blue);
                image.setRGB(x, y, newColor.getRGB());
            }
        }
    }

@Override
    public void handleImage() {
        scaleColor(0.5, 0.5, 1.5);
        setImage1(image);
    }

    public static void main(String[] args) {
        new VideoProcessing();
    }
```

- **Inherits from *VideoGUI***
- **This class's constructor passes title to super's constructor (*VideoGUI*)**
- ***VideoGUI* constructor starts camera and fills *image* instance variable on each shot**
- ***image* instance variable from *VideoGUI* available to this subclass due to inheritance**

- ***handleImage* called every time camera takes a shot, override it here to alter behavior**
- **Calls *scaleColor* to emphasize blue component**

Call *VideoProcessing* constructor on start up

VideoProcessing alters each image taken by the camera before it is displayed

VideoProcessing.java

```
public class VideoProcessing extends VideoGUI {
   public VideoProcessing() {
      super("VideoProcessing");
    }

   public void scaleColor(double scaleR, double scaleG, double scaleB) {
      //safety check
      if (image == null || scaleR < 0 || scaleG < 0 || scaleB < 0) { return; }

      // Nested loop over every pixel
      for (int y = 0; y < image.getHeight(); y++) {
         for (int x = 0; x < image.getWidth(); x++) {
            // Get current color; scale each channel (but don't exceed 255); put new color
            Color color = new Color(image.getRGB(x, y));
            int red = (int)(Math.min(255, color.getRed()*scaleR));
            int green = (int)(Math.min(255, color.getGreen()*scaleG));
            int blue = (int)(Math.min(255, color.getBlue()*scaleB));
            Color newColor = new Color(red, green, blue);
            image.setRGB(x, y, newColor.getRGB());
         }
      }
   }

@Override
   public void handleImage() {
      scaleColor(0.5, 0.5, 1.5);
      setImage1(image);
   }

   public static void main(String[] args) {
      new VideoProcessing();
   }
```

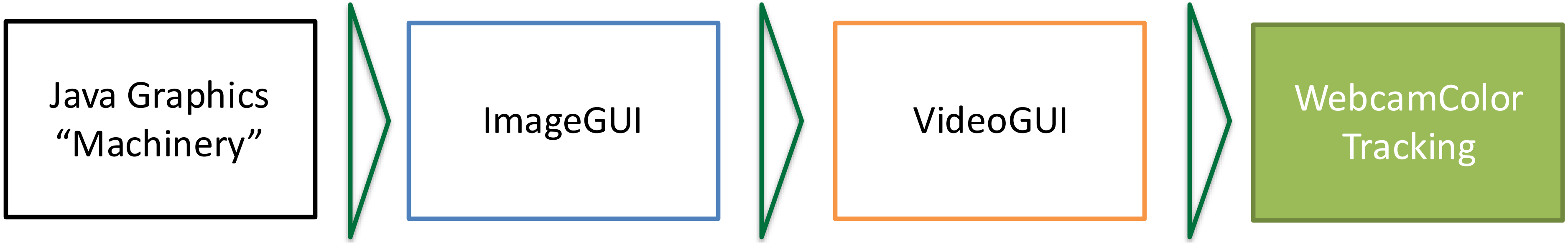
Loop over all pixels in *image*

Scale each color component independently to emphasize blue (don't go over 255!)
Cast double to int

Update *image* pixel with new "bluer" color

Show altered, now "bluer", image on screen instead of the original image captured by the camera

Demo: WebcamColorTracking



Java Graphics “Machinery”

ImageGUI

VideoGUI

WebcamColor Tracking

Notes:

Tracks a color over time

- Click mouse to pick up color from image (use finger tip)
- Will find point with closest color match
- Draws oval around that point as new images arrive (move finger to demonstrate)
- Not too sophisticated, but generally works (Autofocus sometimes causes inaccurate tracking)

WebcamTracking tracks a point from frame to frame

WebcamTracking.java

```
public class WebcamColorTracking extends VideoGUI {
   private Color trackColor=null;      // point-tracking target color instance variable

   /**
    * Constructor, calls super constructor passing title for window
    */
   public WebcamColorTracking() {
      super("WebcamColorTracking");
   }

   <snip>
```

WebcamColorTracking constructor calls super constructor with window title

What is the super class here?

***VideoGUI* – sets up camera, starts taking pictures every 100 ms**

```
   @Override
   public void handleMousePress(int x, int y) {
      System.out.println("Got mouse press");
      if (image != null) {
         trackColor = new Color(image.getRGB(x, y));
         System.out.println("tracking " + trackColor);
      }
   }
```

When mouse is pressed, save the color under the mouse pointer in instance variable *trackColor* (if the camera is running)

```
public static void main(String[] args) {
   new WebcamColorTracking();
}
```

Create object, calls WebcamColorTracking constructor

WebcamTracking tracks a point from frame to frame

WebcamTracking.java

Called when camera takes a shot, override it from VideoGUI to run this code

```
@Override
public void handleImage() {
   super.handleImage();
   if (trackColor != null) {
      // Draw circle at point with color closest to trackColor, then draw circle border in the inverse color
      Point p = track();

      //draw circle around point to highlight
      Graphics g = panel. getWindowReference();
      g.setColor(trackColor);
      g.fillOval(p.x, p.y, 15, 15);
      ((Graphics2D)g).setStroke(new BasicStroke(4)); // thick border
      g.setColor(new Color(255-trackColor.getRed(), 255-trackColor.getGreen(), 255-trackColor.getBlue()));
      g.drawOval(p.x, p.y, 15, 15);
   }
}
```

***super.handleImage* shows *image* instance variable on screen**

Find the closest color to the pixel that was clicked (*track* method code on next slide) Return type of Point

Draw a circle around pixel that most closely matches color

WebcamTracking tracks a point from frame to frame

WebcamTracking.java

Loop over all pixels and return x,y location of pixel with closest color match to *trackColor*

```
private Point track() {
   int cx = 0, cy = 0; // coordinates with best matching color
   int closest = 10000; // start with a too-high number so that everything will be smaller
   // Nested loop over every pixel
   for (int y = 0; y < image.getHeight(); y++) {
     for (int x = 0; x < image.getWidth(); x++) {
       // Euclidean distance squared between colors
       Color c = new Color(image.getRGB(x,y));
       int d = (c.getRed() - trackColor.getRed()) * (c.getRed() - trackColor.getRed())
           + (c.getGreen() - trackColor.getGreen()) * (c.getGreen() - trackColor.getGreen())
           + (c.getBlue() - trackColor.getBlue()) * (c.getBlue() - trackColor.getBlue());

       //track point with closest color to trackColor (so far)
       if (d < closest) {
         closest = d;
         cx = x; cy = y;
       }
     }
   }
   //return point that had the closest color
   return new Point(cx,cy);
}
```

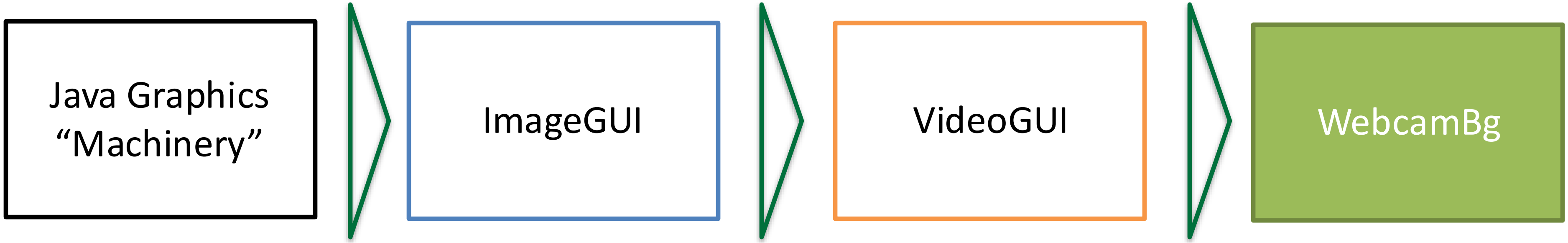
- **Get Color for each pixel**
- **Compare with *trackColor***
- **Save *x,y* with closest color**

Keep track of closest color and its x,y location

Return closest point as variable of type Point

- **Could we just use *Math.abs(c-trackColor)*?**
- **No, because a color is really a 24-bit number**
- **Red is leftmost 8 bits**
- **A 1-bit change in red color would lead to a large difference in *d***

Demo: WebcamBg.java



Notes:

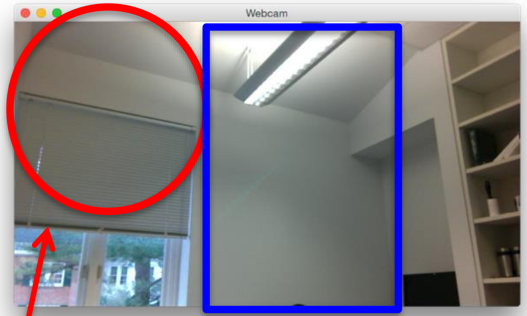
Makes a “green screen” type of effect

- Load a scenery image (Baker tower)
- Click to capture background image from camera
- Now move around
- Compare current and background image color at each x,y location
- If not much color difference, color pixel at x,y with scenery color (e.g., Baker tower)
- Else, color pixel with current image
- Result is you appear to be in front of Baker tower

WebcamBg.java uses three images to make you appear to be somewhere else

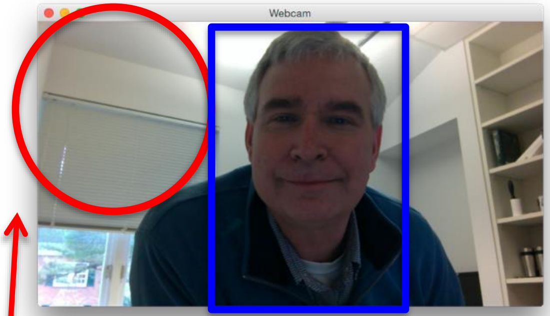
scenery

- Static image
- This is where we want you to appear to be located



background

Static snapshot of the camera's view without you in it



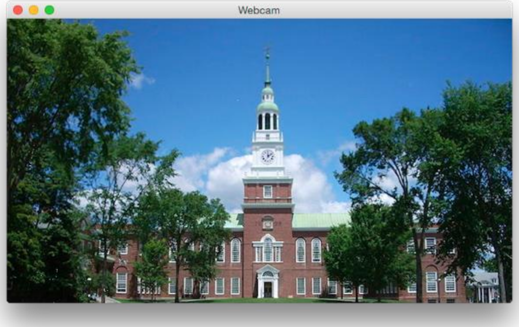
image

Live image as it comes from the camera

- **This portion of the background and live image are the same (mostly)**
- **Show scenery (Baker tower) there**

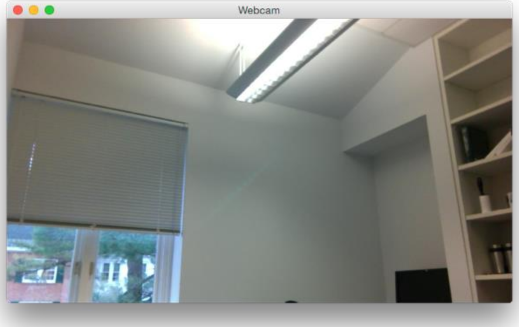
- **This portion of the background and live image are the different**
- **Show live camera image there**

WebcamBg.java uses three images to make you appear to be somewhere else



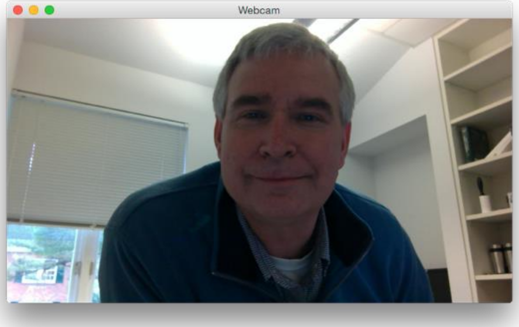
scenery

- Static image
- This is where we want you to appear to be located



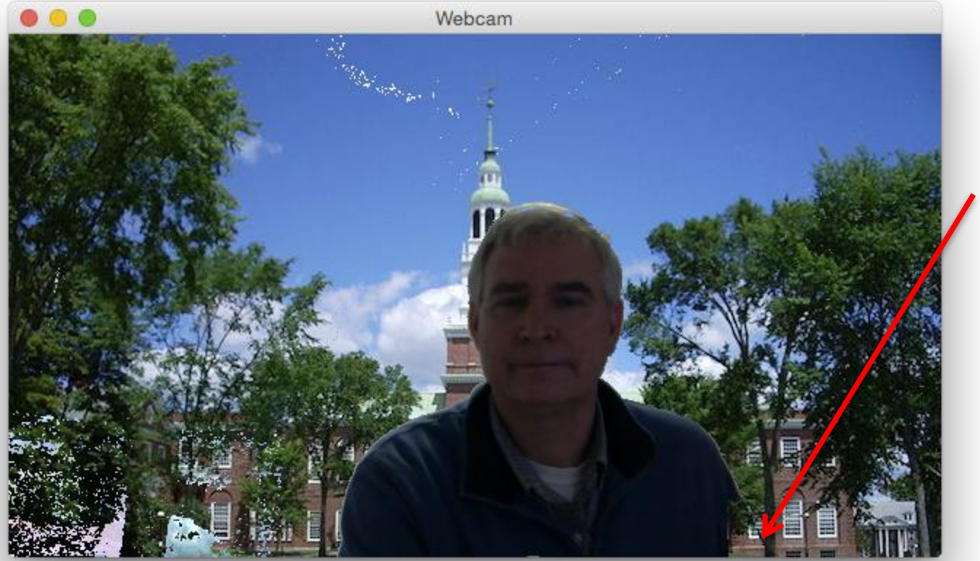
background

Static snapshot of the camera's view without you in it



image

Live image as it comes from the camera



- **Why is this part Baker instead of my arm?**
- **Background is close to my shirt color there**

WebcamBg.java: Replace background with image we choose (green screen effect)

WebCamBg.java

Define threshold, if color difference less than this, use scenery image, else camera image

WebcamBg.java

```
public class WebcamBg extends VideoGUI {
  private static final int backgroundDiff=250;  // setup: threshold for considering a pixel to be background

  private BufferedImage background;    // the stored background grabbed from the webcam
  private BufferedImage scenery;       // the replacement background (e.g., Baker)

  public WebcamBg(BufferedImage scenery) {
    this.scenery = scenery;
  }

  /**
   * VideoGUI method, here setting background as a copy of the current image.
   */
  @Override
  public void handleMousePress(int x, int y) {
    if (image != null) {
      //save background image that we will subtract out
      background = new BufferedImage(image.getColorModel(), image.copyData(null), image.getColorModel().isAlphaPrem
      System.out.println("background set");
    }
  }
```

Load scenery image (Baker tower) to show if small color differences with background image (taken on mouse click)

On mouse press, copy current image as background

WebcamBg.java: Replace background with image we choose (green screen effect)

WebCamBg.java

WebcamBg.java

```
@Override
public void handleImage() {
   if (background != null) {
     // Nested loop over every pixel
     for (int y = 0; y < Math.min(image.getHeight(), scenery.getHeight()); y++) {
       for (int x = 0; x < Math.min(image.getWidth(), scenery.getWidth()); x++) {
         // Euclidean distance squared between colors
         Color c1 = new Color(image.getRGB(x,y));
         Color c2 = new Color(background.getRGB(x,y));
         int d = (c1.getRed() - c2.getRed()) * (c1.getRed() - c2.getRed())
             + (c1.getGreen() - c2.getGreen()) * (c1.getGreen() - c2.getGreen())
             + (c1.getBlue() - c2.getBlue()) * (c1.getBlue() - c2.getBlue());
         //check if distance less than threshold to replace image with scenery, otherwise, keep image
         if (d < backgroundDiff) {
           // Close enough to background, so replace
           image.setRGB(x,y,scenery.getRGB(x,y));
         }
       }
     }
   }
   //update image on screen
   setImage1(image);
}
```

If background is set, loop over each x,y location

Compare color of camera image with *background* image

If not much color difference between current image and *background* image (e.g., no change from background), show *scenery* color for this pixel, else don't change live camera image at this pixel

Key points

1. Images are made up of pixels
2. Each pixel is a Color object
3. Color objects can manipulate red, green, and blue components
4. Video can be thought of as a sequence of images
5. Each image can be altered (just be done before next image arrives)