Graphics & Multimedia

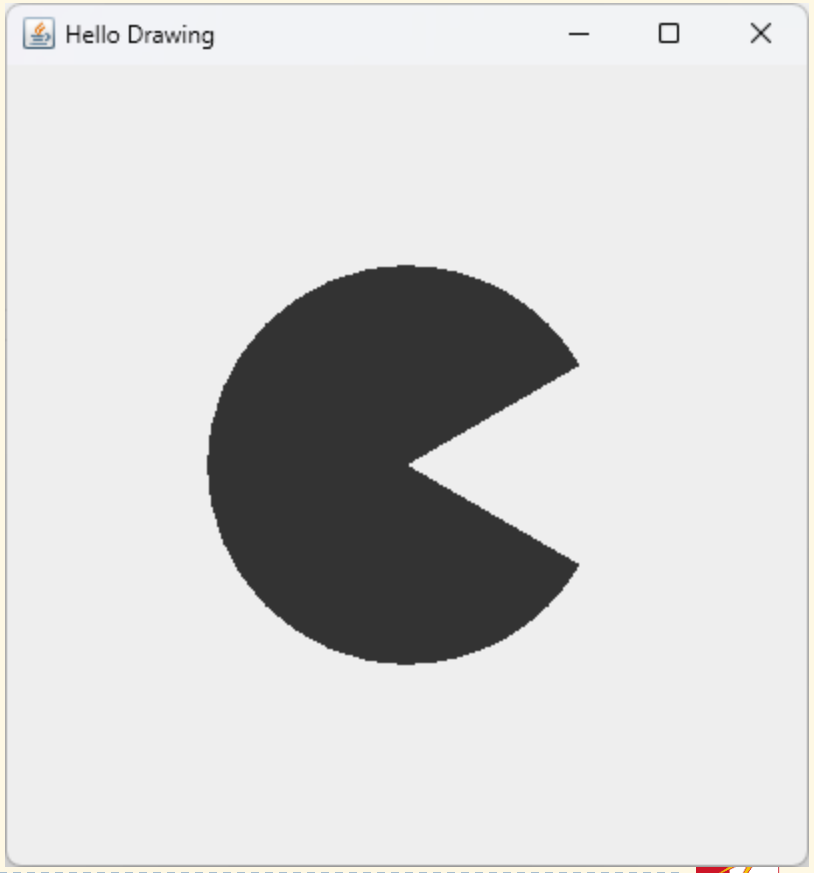
- AWT Canvas
- Animation
- Painting on Swing Components
- Interaction
- Multimedia

AWT Canvas

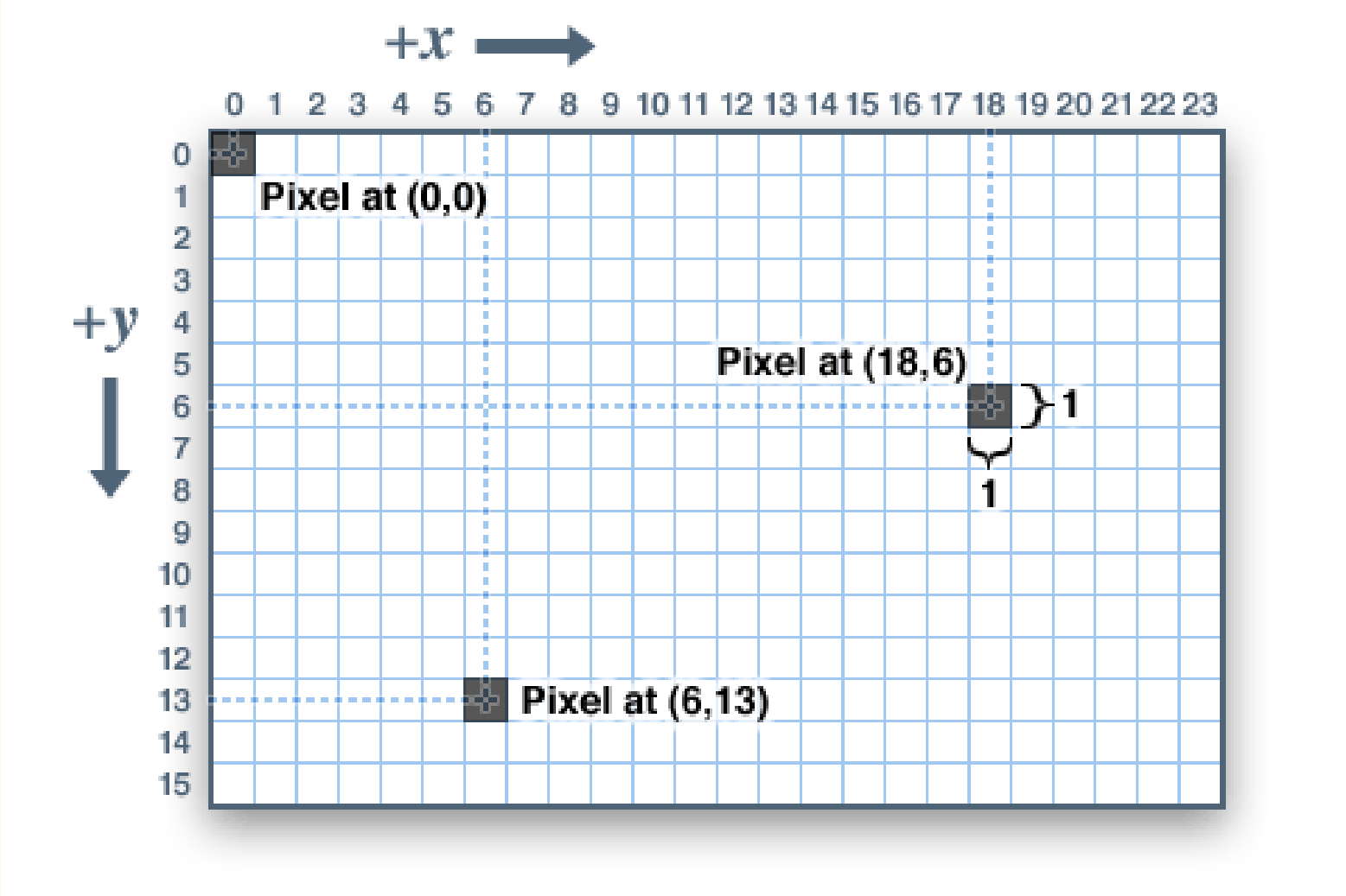
Introduction

- Use AWT `Canvas` to perform custom graphics
  - Override the `paint()` method to on the canvas with a `Graphics` object
- Example:

```
Canvas canvas = new Canvas() {
  @Override
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    g2.fillArc(100, 100, 200, 200, 30, 300);
  }
};
canvas.setSize(400, 400);

frame.add(canvas);
frame.pack();
```



Coordinates



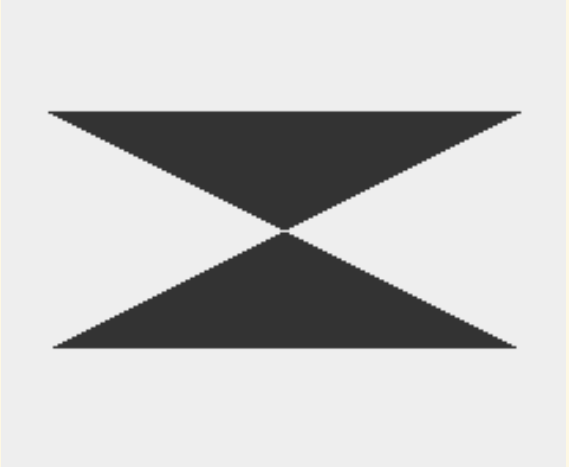
When is `paint()` Called?

- Two kinds of painting operations:
  - System-triggered painting
    - Component is first made visible on the screen
    - Component is resized
    - Component has areas needing to be repainted (e.g., an area that previously hidden by another component, or not in the screen window)
  - Application-triggered painting
    - The application can decide to repaint portions of a component by itself
- Activity: In the last example:
  - Add a printing statement to the `paint()` method to know when it's called
  - Try to resize and move the window around and see what happens!

Auto-fitting

- Scale the drawing proportionally to the canvas size
  - ```
    Graphics2D g2 = (Graphics2D)g;

    // get the current canvas size
    Dimension size = getSize();

    // half of the smaller dimension
    int diameter = Math.min(size.width, size.height) / 2;

    int left = (size.width - diameter) / 2;
    int top = (size.height - diameter) / 2;

    g2.fillArc(left, top, diameter, diameter, 30, 300);
    ```

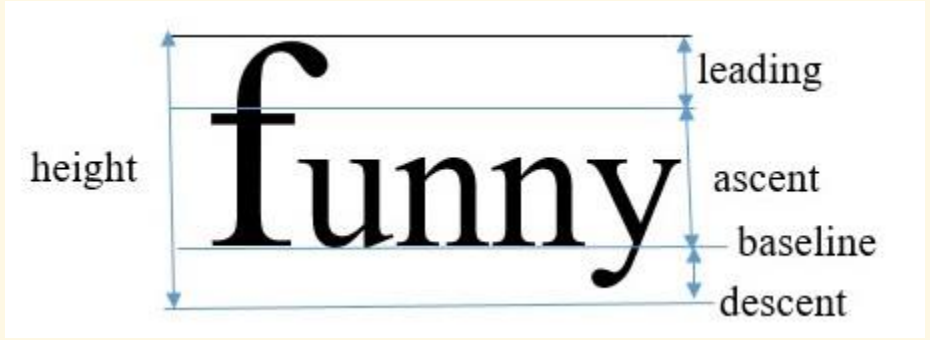
- Activity: Now, try to resize the window again to see the effect
    ```

Drawing Basic Shapes

- Use `drawXyz()` methods to draw the outline of basic shapes:
  - `drawLine(x1, y1, x2, y2)`
  - `drawRect(x, y, width, height)`
  - `drawRoundRect(x, y, width, height, arcWidth, arcHeight)`
  - `drawOval(x, y, width, height)`
  - `drawArc(x, y, width, height, startAng, endAng)`
  - `drawPolyline(xPoints, yPoints, nPoints)`
  - `drawPolygon(xPoints, yPoints, nPoints)`
- Use `fillXyz()` methods to draw filled shapes:
  - `fillRect(x, y, width, height)`
  - `...`

Geometric Primitives

- A more advanced way to draw is to construct geometric primitive objects from the `java.awt.geom` package, then pass into `draw()` or `fill()` methods:
  - `Point2D`, `Line2D`, `Rectangle2D`, `RoundRectangle2D`, `Arc2D`, `Ellipse2D`, `CubicCurve2D`, `QuadCurve2D`
- Example:

```
Line2D line = new Line2D.Double(10, 50, 100, 200);
Ellipse2D ellipse = new Ellipse2D.Double(100, 100, 50, 60);

g2.draw(line);
g2.draw(ellipse);
```

Drawing Arbitrary Shapes

- To create more complicated geometry, use `GeneralPath` which represents a geometric path constructed from lines, and quadratic and cubic curves
- Example:

```
GeneralPath path = new GeneralPath(GeneralPath.WIND_EVEN_ODD);
path.moveTo(100, 100);
path.lineTo(300, 200);
path.lineTo(100, 200);
path.lineTo(300, 100);
path.closePath();
g2.fill(path);
```



Stroke Settings

- Include:
  - Pen width: pixels measured perpendicularly to the pen trajectory
  - End cap type: `CAP_BUTT`, `CAP_ROUND`, `CAP_SQUARE`
  - Line join type: `JOIN_BEVEL`, `JOIN_MITER`, `JOIN_ROUND`
  - Miter limit: the limit to trim a line join using `JOIN_MITER` decoration
  - Dash pattern and phase
- Example:

```
BasicStroke stroke = new BasicStroke(3,
    BasicStroke.CAP_ROUND,
    BasicStroke.JOIN_MITER, 5,
    new float[]{10, 10, 1, 10}, 0);
g2.setStroke(stroke);
g2.drawLine(100, 100, 200, 200);
```

Stroke and Fill Patterns

- Create an instance of an object that implements the `Paint` interface and pass it into `setPaint()` method
  - 3 predefined classes: `Color`, `GradientPaint`, and `TexturePaint`
- Examples:
  - ```
    g2.setPaint(Color.GREEN);
    g2.drawLine(100, 100, 200, 200);
    ```
  - ```
    Paint gradient = new GradientPaint(0, 0, Color.RED, 10, 10, Color.GREEN, true);
    g2.setPaint(gradient);
    g2.drawLine(100, 100, 200, 200);
    ```
  - ```
    // image loading should not be put inside paint()
    BufferedImage img = null;
    URL url = HelloGraphics.class.getClassLoader().getResource("image/pattern.png");
    img = ImageIO.read(url);

    Rectangle2D anchor = new Rectangle(0, 0, img.getWidth(), img.getHeight());
    g2.setPaint(new TexturePaint(img, anchor));
    g2.fillOval(100, 100, 100, 100);
    ```
    ```

Draw Texts

- Left-bottom aligned text:
  - ```
    Font font = new Font("Fancy",
        Font.BOLD | Font.ITALIC, 100);
    g2.setFont(font);
    g2.drawString(text, x0, y0);
    ```



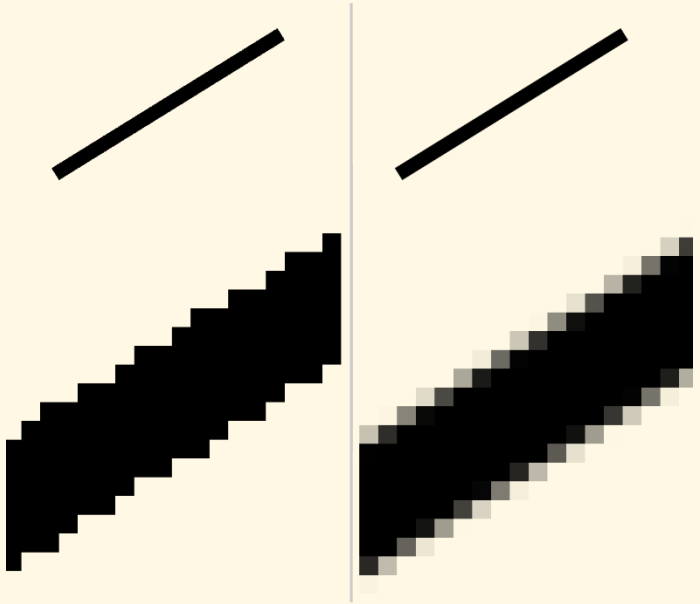
- Center-aligned text:
  - ```
    FontMetrics metrics = g2.getFontMetrics();
    int x = x0 - metrics.stringWidth(text) / 2,
        y = y0 - metrics.getHeight() / 2
            + metrics.getAscent();
    g2.drawString(text, x, y);
    ```

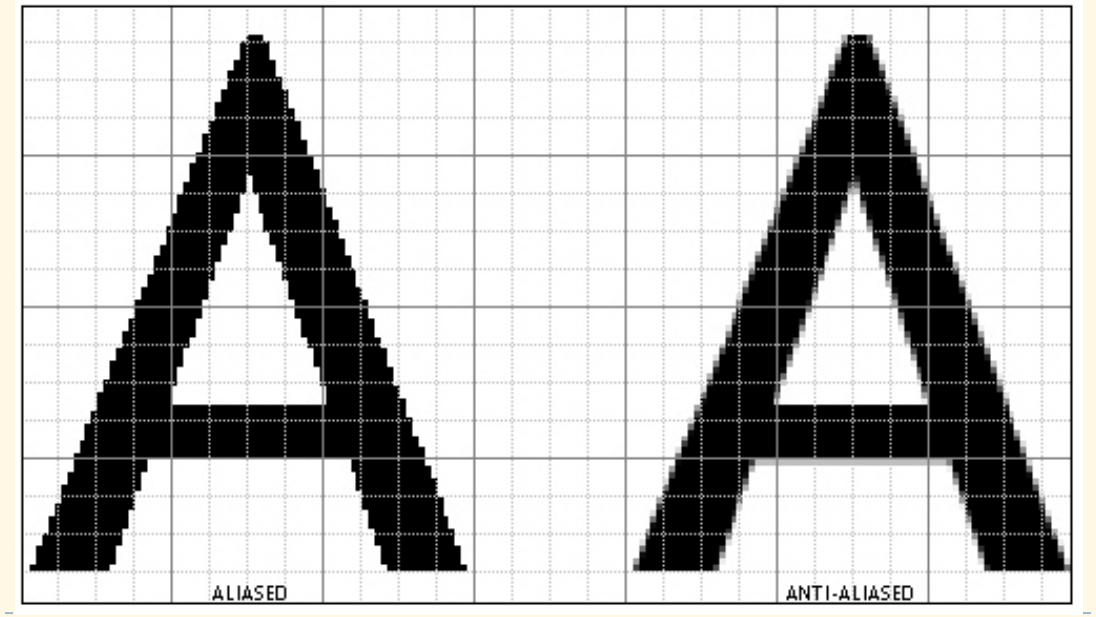
Draw Images

```
Paint gradient = new GradientPaint(
    0, 0, Color.YELLOW,
    300, 300, Color.GREEN);
g2.setPaint(gradient);
g2.fillOval(50, 50, 300, 300);

InputStream imgStream = HelloGraphics.class
    .getResourceAsStream("image/mcqueen.png");
BufferedImage image = ImageIO.read(imgStream);

g2.scale(0.3, 0.3);
g2.drawImage(image, 0, 250, null);
```

Anti-aliasing

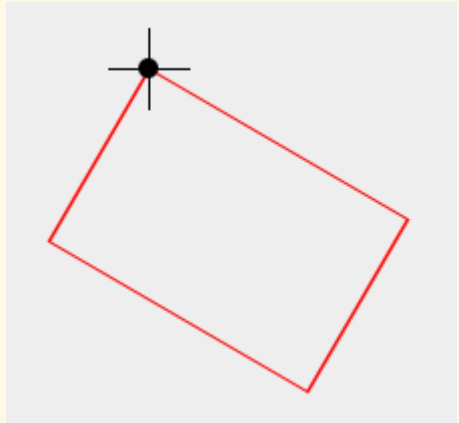
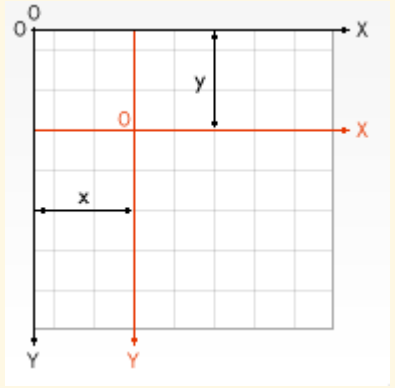
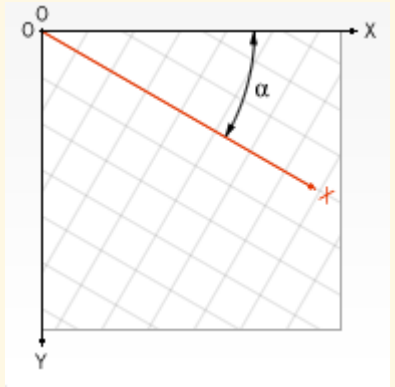
- Using resampling techniques to reduce the problems of aliasing and obtain better rendering quality:
  - ```
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
    RenderingHints.VALUE_ANTIALIAS_ON);
    g2.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
    RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
    ```




    ```

Transforms

- Coordinates can be transformed with `translate()`, `rotate()`, `scale()`, `shear()` methods
- Alternatively, pass a `AffineTransform` object into `transform()` or `setTransform()` methods
- Example:
  - ```
    AffineTransform oldTransform = g2.getTransform();

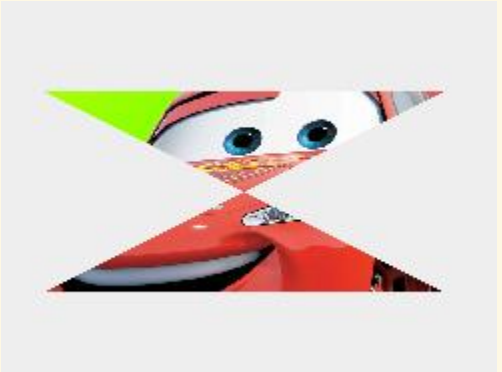
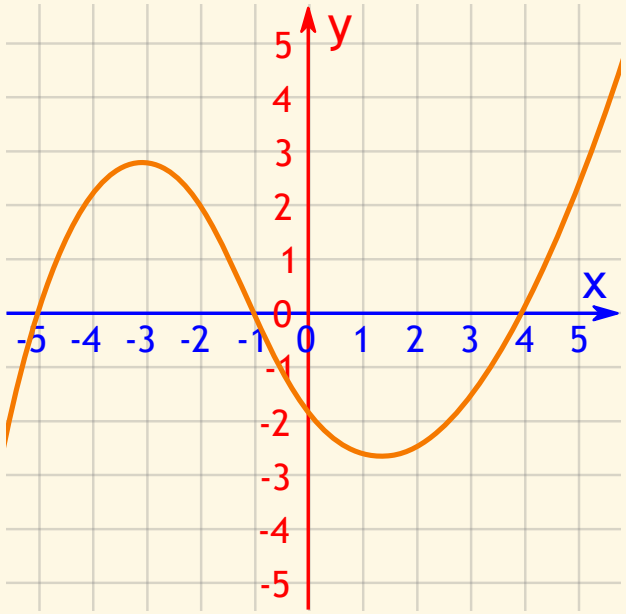
    g2.translate(100, 100);
    g2.rotate(30. * Math.PI / 180.);
    g2.scale(1.5, 1.);
    g2.drawRect(0, 0, 100, 100);

    g2.setTransform(oldTransform);

    // continue to draw other stuffs:
    // ...
    ```





translate()



rotate()
    ```

Clipping

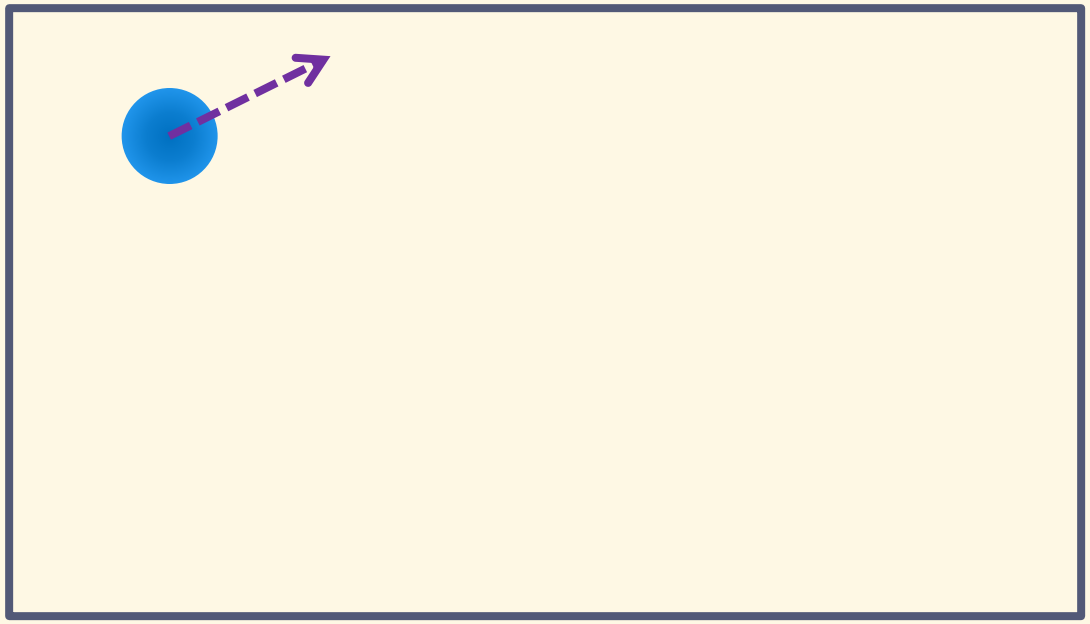
- Removing parts of elements defined by other parts
  - Any `Shape` object can be passed into `setClip()` method to be the clipping path
- Example:
  - ```
    GeneralPath clip = new GeneralPath(GeneralPath.WIND_EVEN_ODD);
    clip.moveTo(100, 100);
    clip.lineTo(300, 200);
    clip.lineTo(100, 200);
    clip.lineTo(300, 100);
    clip.closePath();
    g2.setClip(clip);
    ```


    ```

Pixel Operation Function

- It specifies how new pixels are to be combined with the existing pixels on the graphics device during the rendering process
- Methods to choose the mode:
  - `setComposite(Composite comp)`: the generic method to use the given function
    - See the `AlphaComposite` for implementations of the `Composite` interface
  - `setPaintMode()`: set to override mode
    - Equivalent to `g2.setComposite(AlphaComposite.SrcOver)`
  - `setXORMode(Color c)`: set to XOR mode
    - Equivalent to `g2.setComposite(AlphaComposite.Xor)`

Exercises

1. Draw a graph of math functions (including axes, grid, labels,…)
2. Draw a static analog clock face showing the current time



Animation

Repainting

- One may need to refresh the drawing in a Canvas to reflex the content updates
- Example:
  - Think about a Swing app using a `Canvas` to show a 4-digit PIN which changes every a given number of seconds by a Swing `Timer`:
  - Problem: How to refresh the canvas when the PIN gets update?

`repaint()` Method

- To refresh the canvas with new contents, do not call `paint()` method directly, but call `repaint()` method instead, because:
  - The refreshing process is not just invoking `paint()` method. The exact invoked method is `update()`, which erase the background and calls `paint()`
  - `repaint()` is asynchronous: It posts a repaint request to the event queue, so the repaint may occur later when the `RepaintManager` is free enough
  - The `RepaintManager`, for efficiency, usually collapse repaint requests into one if multiple of them are in the event queue
- The code to refresh the canvas (full code in `PinNumber.java`):

```
new Timer(10000, event -> {
    pin = generateNewPin();
    canvas.repaint();
}).start();
```

Refresh Flickering Problem

- As mentioned, the default `update()` method clears the canvas before invoking `paint()` method
  - This causes the flicker effect when the canvas refreshes at a high rate
  - See the `PacmanFlickering.java` example

- Solution:
  - Override the `update()` method to skip the default behavior
  - Use the so-called “double buffering” technique:
    - Paint the new drawing to a back buffer
    - Push the result from the back buffer to the front canvas when needed

Flicker-Free Solution

```
Canvas canvas = new Canvas() {
    private Image bufferImage = null;

    @Override
    public void update(Graphics g) {
        paint(g);
    }

    @Override
    public void paint(Graphics g) {
        if (bufferImage == null) bufferImage = createImage(400, 400);
        Graphics2D bufferGr = (Graphics2D)bufferImage.getGraphics();
        bufferGr.setBackground(Color.GRAY);
        bufferGr.clearRect(0, 0, 400, 400); // clear old contents

        // draw new contents

        g.drawImage(bufferImage, 0, 0, this);
    }
};
```

- See full solution in `PacmanFlickerFree.java`

Exercises

Write programs for:

1. An analog clock which runs, and make sure that it's flicker free
2. A ball bouncing within a rectangle



Painting on Swing Components

Overview

- It’s possible to extend any Swing component to change its default appearance:
  - Override the `paintComponent()` method
  - Choose to call or skip `super.paintComponent()` as desired
  - Calling `update()` is not necessary

Custom `JLabel`

- Example (see full code in `SwingPacman.java`):
  - ```
    class MyCanvas extends JLabel {
        @Override
        public void paintComponent(Graphics g) {
            Graphics2D g2 = (Graphics2D)g;
            // ...
        }
    }
    ```
    ```

Custom `JButton`

- The `model` field (instance of `ButtonModel` class) of `JButton` class has methods to get the button state: `isPressed()`, `isRollover()`, `isEnabled()`
- Example (full code in `CustomSwingButton.java`):
  - ```
    Color bgColor = model.isPressed() ? Color.BLUE :
        model.isRollover() ? Color.LIGHT_GRAY : Color.GRAY;
    Color textColor = model.isPressed() ? Color.WHITE : Color.BLACK;

    g2.setPaint(bgColor);
    g2.fill3DRect(0, 0, dim.width, dim.height, !model.isPressed());

    g2.setFont(font);
    FontMetrics fontMetrics = g2.getFontMetrics();
    int x = dim.width / 2 - fontMetrics.stringWidth(getText()) / 2,
        y = dim.height / 2 + 5;
    g2.setPaint(textColor);
    g2.drawString(getText(), x, y);
    ```
    ```

Exercise

- Implement a custom Swing checkbox

Interaction

Overview

- GUI apps may need to allow the user to interact in different ways:
  - Using buttons or other components
  - Using mouse
  - Using keyboard
  - …

Mouse Events

- Use `MouseListener` interface for mouse click, press, enter, exit events
  - `void mousePressed(MouseEvent e)`
  - `void mouseReleased(MouseEvent e)`
  - `void mouseClicked(MouseEvent e)` (pressed then released)
  - `void mouseEntered(MouseEvent e)`
  - `void mouseExited(MouseEvent e)`
- Use `MouseMotionListener` interface for mouse motion events
  - `void mouseMoved(MouseEvent e)`
  - `void mouseDragged(MouseEvent e)`
- Use `MouseWheelListener` interface for mouse wheel related events
  - `void mouseWheelMoved(MouseWheelEvent e)`
- Use `MouseAdapter` abstract class which implements all 3 above classes for any subset of the events

Example

- See `ClickForDots.java` for the full code
  - ```
    JLabel canvas = new JLabel() {
        @Override
        public void paintComponent(Graphics g) {
            Graphics2D g2 = (Graphics2D)g;
            g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);

            g2.setPaint(Color.BLUE);
            for (Point2D p : dots)
                g2.fillOval((int)p.getX() - 5, (int)p.getY() - 5, 10, 10);
        }
    };

    canvas.addMouseListener(new MouseInputAdapter() {
        @Override
        public void mousePressed(MouseEvent evt) {
            dots.add(new Point2D.Double(evt.getX(), evt.getY()));
            canvas.repaint();
        }
    });
    ```
    ```

Hit Testing

- IIs determining if we have clicked inside a shape with a mouse pointer
  - Each AWT `Shape` instance has a `contains()` method to support this task
- Example (full code in `PacmanHitTest.java`):
  - ```
    public void mousePressed(MouseEvent evt) {
        Point2D objCoords = new Point2D.Double(
            evt.getX() - center.getX(),
            evt.getY() - center.getY());

        if (shape.contains(objCoords)) {
            hitting = true;
            canvas.repaint();
        }
    }
    ```
    ```

Key Events

- Key events indicate when the user is typing at the keyboard
  - Fired by the component with the keyboard focus when the user presses or releases keyboard keys
  - To fire keyboard events, a component must have the keyboard focus
    - Some components like `JLabel` are not focusable by default, one can make it focusable by calling `setFocusable(true)`
  - Use `KeyListener` interface or `KeyAdapter` abstract class to handle these events
    - `void keyTyped(KeyEvent e)`
    - `void keyPressed(KeyEvent e)`
    - `void keyReleased(KeyEvent e)`

Example

- See `SwingPacmanWithKeys.java` for the full code
  - ```
    // listen from the frame, not the canvas
    frame.addKeyListener(new KeyAdapter() {
        @Override
        public void keyPressed(KeyEvent e) {
            if (e.getKeyCode() == KeyEvent.VK_RIGHT) {
                center.setLocation(center.getX() + 10,  center.getY());
                canvas.repaint();
            } else if (e.getKeyCode() == KeyEvent.VK_LEFT) {
                center.setLocation(center.getX() - 10,  center.getY());
                canvas.repaint();
            }
        }
    });
    ```
    ```

Exercises

1. Write a program that allows the user to draw lines (or circles) with the mouse
2. Modify the `PacmanHitTest.java` example to allow the user to drag the shape and move it around with the mouse

Multimedia

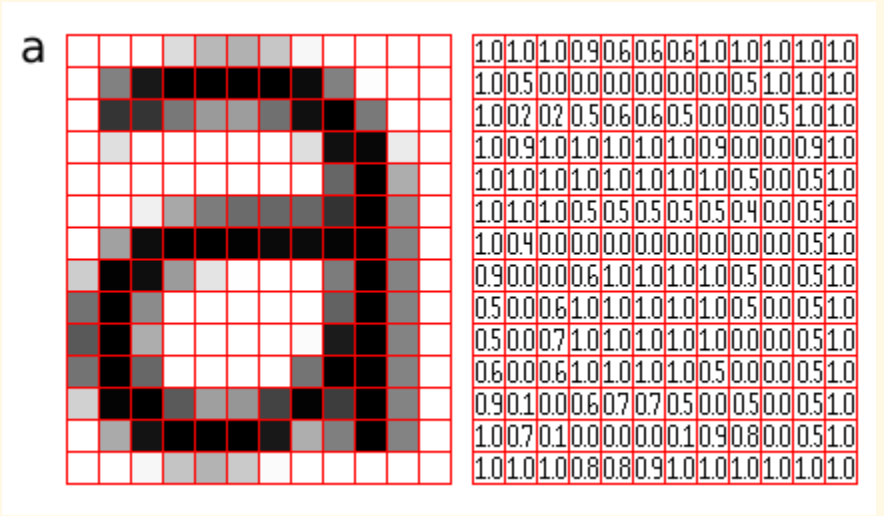
Overview

- Want to play audio, video contents?
- Existing solutions:
  - Java Sound (`javax.sound` package): audio playback
  - Java Media Framework (JMF - `javax.media` package): video and audio playback, but almost obsolete

Java Sound API

- Using Java Sound (`javax.sound` package)
- Example (full code in `SoundPlayer.java`):
  - ```
    try (
        InputStream inputStr = SoundPlayer.class.getClassLoader()
            .getResourceAsStream("media/sample-sound.wav");
        AudioInputStream audioStr = AudioSystem
            .getAudioInputStream(inputStr);
        Scanner scanner = new Scanner(System.in)
    ) {
        Clip audioClip = AudioSystem.getClip();
        audioClip.open(audioStr);
        audioClip.start();
        //...
    }
    ```
    ```

Video Playback using vlcj

- vlcj:
  - Open-source project to allow an instance of a native VLC media player to be embedded in a Java application
  - https://github.com/caprica/vlcj
- Setting up:
  - Download the latest stable version of vlcj and add to the project
    - https://capricasoftware.co.uk/projects/vlcj-4/tutorials/installation
    - `vlcj` jar file along with the `vlcj-natives`, `jna` and `jna-platform` jar files that vlcj depends on

Example

- See full code in `VideoPlayer.java`:

```
import javax.swing.*;
import uk.co.caprica.vlcj.player.component.EmbeddedMediaPlayerComponent;

public class VideoPlayer {
    public static void main(String argv[]) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Video Player");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(600, 500);
            frame.setVisible(true);

            EmbeddedMediaPlayerComponent mediaPlayerComponent =
                new EmbeddedMediaPlayerComponent();
            frame.setContentPane(mediaPlayerComponent);
            mediaPlayerComponent.mediaPlayer().media()
                .play("media/sample-video.mpg");
        });
    }
}
```

Image Processing

Introduction

- Image processing is the process of transforming an image into a digital form and performing certain operations to get some useful information from it
- Basic operations:
  - Geometrical transforms: scaling, rotating, cropping,…
  - Color transforms: lightening/darkening, increasing/decreasing contrast,…

Image Representation

- A *bitmap* (or raster) image represents a natural image in the form of an 2D array (or matrix), where the value of each element, called a *pixel*, corresponds to the color of that portion of the image
- Color depth: number of bits per pixel
- Color spaces:
  - Binary (B/W)
  - Grayscale
  - Color: RGB, HSV,…
    - Color channels
- The most used: RGB, grayscale



Image I/O

- Use `BufferedImage` class from `java.awt.image` package
- Reading:
  - ```
    File imgFile = new File("sample.jpg");
    BufferedImage img = ImageIO.read(imgFile);
    ```
- Writing:
  - ```
    File imgFile = new File("output.jpg");
    ImageIO.write(img, "png", imgFile);
    ```

Elementary Operations

- Getting image information:
  - `int width = img.getWidth();`
  - `int height = img.getHeight();`
  - `ColorModel colorModel = img.getColorModel();`
  - `int type = img.getType();`
- Getting/setting a pixel:
  - `int rgb = img.getRGB(50, 50);`
  - `img.setRGB(50, 50, rgb);`
- Extracting a sub-image:
  - `BufferedImage subimg = img.getSubimage(20, 10, 50, 50);`

Example: Convert an RGB Image to Grayscale

```
for (int i = 0; i < img.getWidth(); i++)
    for (int j = 0; j < img.getHeight(); j++) {
        int rgb = img.getRGB(i, j),
            r = (rgb >> 16) & 0xff,
            g = (rgb >> 8) & 0xff,
            b = rgb & 0xff;

        int avg = (r + g + b) / 3;
        rgb = (avg << 16) | (avg << 8) | avg;

        img.setRGB(i, j, rgb);
    }
```

Drawing

- Get a `Graphics` object from a `BufferedImage` object and use it for drawing operations:
  - ```
    Graphics2D bufferGr = (Graphics2D)img.getGraphics();

    // reset with a new background
    bufferGr.setBackground(Color.GRAY);
    bufferGr.clearRect(0, 0, 400, 400); // clear old contents

    // draw new contents...
    ```
    ```

Resizing (aka Scaling)

```
int newWidth = 400,
    newHeight = 500;

Image scaledImg = img.getScaledInstance(
    newWidth, newHeight, Image.SCALE_SMOOTH);

BufferedImage outputImg = new BufferedImage(
    newWidth, newHeight, BufferedImage.TYPE_INT_RGB);
outputImg.getGraphics().drawImage(scaledImg, 0, 0, null);
```

Exercises

Write programs to:

- Flip an image horizontally and vertically
- Lighten or darken an image
- Add a given watermark to an image
- Crop an image to a given size
- Rotate an image with a given angle