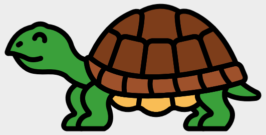
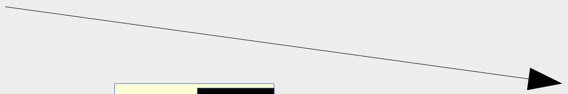
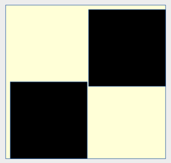
ICS 111
Drawing

- review: javax.swing.JComponent
- review: java.awt.Graphics, rectangles, ovals, lines
- text fields and text areas
- turtle graphics
- vector graphics
- bitmap graphics







review: javax.swing.JComponent

- JComponent is an abstract class with many useful subclasses, including:
  - AbstractButton, the superclass for JButton, JToggleButton, and JMenuItem
  - JPanel
  - JLabel
  - JOptionPane
- JComponent has a protected method

  ```
  void paintComponent(Graphics g)
  ```

  that subclasses may override
- paintComponent is called when the frame that a component has been added to, is displayed or resized

review: java.awt.Graphics

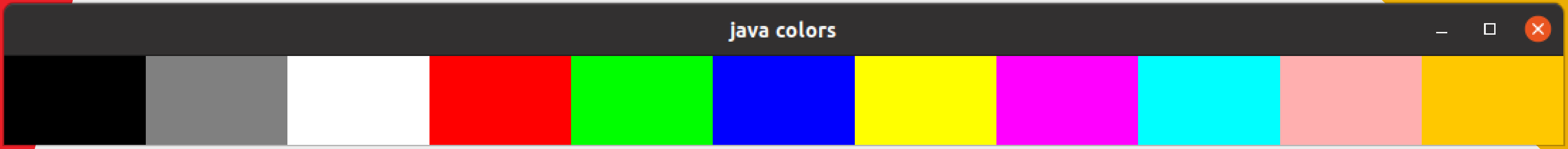
- Graphics is the main class for drawing in awt and swing
  - designed for bitmap graphic painting
- for normal drawing applications, we do not directly create a new Graphics object, instead we use the one given as the parameter to paintComponent
  - or can call the component’s getGraphics()
- painting operations use the current color, current paint mode (XOR or Paint), and current font

```
void setColor(Color c)
```

-

java.awt.Color

- Color defines colors using Red, Green, and Blue (RGB) values
- each of the RGB values goes from 0 (dark) to 255 (bright)
  - e.g. 0,0,0 is black, 128,128,128 is gray, 255,255,255 is white
- Color also pre-defines many colors
- including Color.BLACK, Color.GRAY, Color.WHITE
- and also Color.RED, Color.GREEN, Color.BLUE, Color.YELLOW, Color.MAGENTA, Color.CYAN, Color.PINK, Color.ORANGE



java.awt.Color example

```
import java.awt.Color;

public class Colors extends javax.swing.JFrame {
  public Colors() {  // constructor
    final Color[] myColors = { Color.BLACK, Color.GRAY, Color.WHITE,
                               Color.RED, Color.GREEN, Color.BLUE,
                               Color.YELLOW, Color.MAGENTA, Color.CYAN,
                               Color.PINK, Color.ORANGE };
    // JFrame operations
    setTitle("java colors");
    setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
    setSize(myColors.length * 100, 100);
    // add a single JComponent which paints all the color blocks
    add(new javax.swing.JComponent() {
      protected void paintComponent(java.awt.Graphics g) {
        int offset = 0;  // where to put the color block, horizontally
        for (Color c: myColors) {
          g.setColor(c);
          g.fillRect(offset, 0, 100, 100);
          offset += 100;
        }
      }
    });
    setVisible(true);
  }
  // open a window and display our predefined colors
  public static void main(String[] a) {
    Colors colorsObject = new Colors();
  }
}
```

review: rectangles, ovals, lines

- these are primitives in java.awt.Graphics
- fillRect and drawRect take as parameters: x, y, width, height
  - as is conventional for graphics, y starts at the top of the window and grows downwards
- fillOval and drawOval take the same parameters as defining the bounding box of the oval
  - a circle has width == height
- drawLine takes as parameters the x and y of the two endpoints
  - drawLine(0, 0, 100, 100) draws a diagonal line
  - lines always have thickness 1

drawing text and images

- drawString(String text, int x, int y)
- the x,y position is the lower left position of the baseline of the text
  - letters such as g, j, p, q, y may extend below the baseline
- the text is drawn in the current color and current font
- the font is defined as a java.awt.Font
- a font always has a size
- a font has a style, such as plain, bold or italic
- Java has five logical fonts which should be available in any Java implementation. Portable Java code should only use one of these:
  - Serif, SansSerif, Monospaced, Dialog, DialogInput
- alternatively, java.awt.GraphicsEnvironment.getAllFonts() returns all fonts available in the current environment
- java.awt.Graphics.drawImage is similar to fillRect, but takes an image, possibly scaling it to fit

text fields

- the JTextField class allows the user to enter text

```
int width = 50; // a 50-character field
javax.swing.JTextField tf =
  new javax.swing.JTextField(width);
...
// in actionPerformed:
String value = tf.getText();
```

- the field is limited to a single line
- the field is unlabeled – use a JLabel to add information
  - it is possible to set a tooltip

text areas

- unlike a text field, a JTextArea allows multiple lines
  - the constructor takes a number of rows and a number of columns:

```
JTextArea ta = new JTextArea(10, 80); // 10 lines
of 80 characters
```

- newlines – "\n" – indicate the end of a line and the start of a new
- text areas may be editable or read-only

```
ta.setEditable(false);
```

  - text fields can also be set read-only
- the code can still change a text area that is not editable by the user
-

scroll panes

- a text area, or any other Component, can be decorated with scroll bars by creating a JScrollPane with that component as the argument to the constructor

```
JTextArea ta = . . .
JScrollPane sp = new JScrollPane(ta);
panel.add(sp);
```

- adding the scroll pain makes the text area available, with scroll bars around it

turtle graphics: history

- in the 1960s, the Logo language was created to teach children how to program
- part of the fun was to program a mechanical turtle to move around a large sheet of paper
- wherever the turtle put its pen down, it would create part of a drawing
- not everybody owns large sheets of paper and a mechanical turtle, so turtle graphics can also be used on a screen

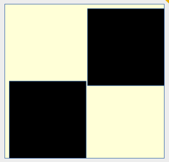
turtle graphics: principles

- at any given time, the turtle has a position and direction
- turtle commands include:
  - turn a given number of degrees
  - move a certain number of steps with the pen down
  - forward a certain number of steps with the pen up
  - resize the step
- notice that no coordinates are needed!
- straight lines are drawn by moving a given number of steps with the pen down
- curves are drawn by moving a small distance, then turning a small angle, and repeating in a loop
  - to draw a spiral, increase the step size each time around the loop
- turtle graphics make it easy to draw curves that are harder in other systems
  - and polygons are easy if we know all the angles and lengths
  - but filling an area with turtle graphics is slow
- Python has a turtle graphics library
  - Java does not have a standard turtle graphics library

vector graphics

- review: some of the earliest computer graphic displays were vector based
- steerable electronic beams in a cathode ray tube can easily be made to draw straight lines
  - e.g. the lines in an old TV
- curves are harder but possible
- graphic primitives (e.g. drawLine and drawArc in java.awt.Graphics) are often inspired by what can be done with vector graphics
- in vector graphics, outlines are easy, filling areas is very expensive and possibly (depending on the medium) not very satisfactory

bitmap graphics



- review: some computers have used bitmap graphics since the Xerox Alto in the 1970s
- a bitmap is an array of bits
- some bitmaps are automatically displayed on a screen by specialized hardware
  - most modern display devices and printers use bitmap graphics
- fillRectangle and copyRectangle are the basic bitmap graphic operations
  - to render a character in a font, copy a rectangle from the font bitmap to the displayed bitmap
  - Graphics.drawImage copies a rectangle from the image bitmap to the displayed bitmap
- some operations are pixel based, for example, drawing a line
  - setting a pixel is like filling a 1x1 rectangle
  - if you look at a slanted line very closely, you can see it is jagged
- some operations, such as image resizing, are harder and require pixel-by-pixel computations

Summary

- the paintComponent method of JComponent gives us a Graphics object that allows us to draw simple geometric shapes, text, and images
- when working with a JFrame or JPanel, can have text fields or text areas
- we can encapsulate any JComponent in a JScrollFrame
- bitmap graphics is very common, but is complemented by other graphics programming styles, including vector graphics and turtle graphics