ICS 111
Swing

- top-level containers
- z order
- text components
- Point, Dimension, and Rectangle
- size, bounds, preferred size, and packing
- swing and threads
- lambda expressions
- javax.swing.Timer and animation

review: javax.swing.JFrame

- review: JFrame is the object (window) that ultimately gets displayed
- there are three possible top-level objects: JFrame, JDialog, and JApplet
  - JDialog is used for dialogs
  - JApplet was used for Java execution in web browsers
    - JApplet is now deprecated and not generally supported by browsers
- at initialization time, it is a good idea to call `setSize`, `setDefaultCloseOperation(EXIT_ON_CLOSE)`, and finally `setVisible`

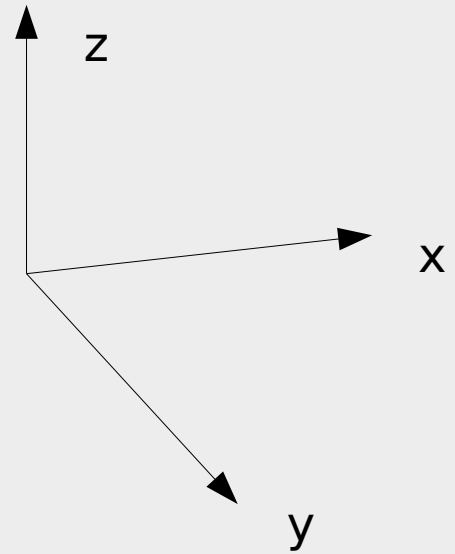
default panes in a JFrame

- by default a JFrame has several panes
- In order:
  - rootPane (the lowest of the panes)
  - layeredPane
  - contentPane (and menu bar)
    - any menu bar included within the frame is displayed next to the contentPane
    - setJMenuBar
  - glassPane (the top pane)
  - see this tutorial
- anything drawn on a later/higher pane covers anything drawn on an earlier/lower pane
  - the layeredPane itself provides several layers, again with later layers covering earlier layers
    - frame content layer has the content pane and the menu bar (-30,000)
    - default layer holds contents added without a depth (0)
    - ...
    - popup layer is for menus and other popups (300)
    - drag layer is for feedback while dragging (400)
  - see this tutorial

z order

- sometimes we want to carefully place graphical objects next to each other
- other times, we just want to cover whatever is in a given position
- covering is done by specifying z order
- lower z covers higher z
- setComponentZOrder

  in java.awt.Container
- or create a JLayeredPane



text components

- review: JTextField and JTextArea
- subclasses of JTextComponent:
- unstyled: JTextField, JFormattedTextField, JPasswordField, JTextArea
  - formatted text field allows you to specify which characters are allowed where and to localize currency and date printing
- styled: JEditorPane, JTextPane
  - may include pictures, varying fonts, etc.
- see this tutorial
-

displaying web pages

```
public class Html extends javax.swing.JFrame {
  final static String HOME_PAGE = "http://www.alnt.org";
  public Html(String url) {
    java.awt.Component component = null;
    try {
      javax.swing.JEditorPane page = new javax.swing.JEditorPane(url);
      page.setEditable(false);
      component = page;
    } catch (Exception e) {
      System.out.println("got exception " + e);
      component = new javax.swing.JLabel(url + ": error " + e);
    }
    javax.swing.JScrollPane scroll = new javax.swing.JScrollPane(component);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    add(scroll);
    pack();  // reset the size to match the contents - no need for setSize
    setVisible(true);
  }
  public static void main(String[] args) {
    if (args.length == 0) {
      Html myPage = new Html(HOME_PAGE);
    } else {
      for (String s: args) {
        Html myPage = new Html(s);
      }
    }
  }
}
```

Point, Dimension, and Rectangle

- a number of primitive dimensional types are defined in java.awt, and used throughout awt and swing
- Point represents an x,y coordinate
- Dimension represents a width and a height
- Rectangle has the x,y of the upper left corner and a width and a height

size and preferred size, pack, bounds

- setSize sets the current size of a JFrame:
  - setSize(int width, int height);
  - method inherited from java.awt.Window
- for any java.awt.Component, can call setPreferredSize(Dimension size)
- java.awt.Window.pack() sets the size “to fit the preferred size and layout of the subcomponents”
  - so JFrame.pack() automatically sets the size
- for any java.awt.Window can call

  setBounds(Rectangle newBounds)

  these bounds give the new position (on the screen) and size of the window

single-threaded model

- when you create a JFrame, the code connected with the JFrame executes in response to user actions, and independently of your main method
- that means the display code runs in a special thread called the **event dispatching thread**
- so now your program has two threads: the main thread and the event dispatching thread
- when the main thread wants to execute something in the event dispatching thread, it must request that the code be invoked later:

```
javax.swing.SwingUtilities.invokeLater(Runnable r)
```

- the Runnable interface requires the run method:

```
public interface Runnable {
  void run();
}
```

- the run method, when scheduled with invokeLater, is allowed to modify frames, repaint, and do any other kind of display operations

lambda expressions

- we have seen many cases of interfaces which only require one method
- these interfaces often are the type of a parameter to a method
  - we may build them using anonymous classes
  - Runnable is an example
- in mathematics, a functional expression is called a **lambda expression** (λ-expression)
- in Java, we can build an anonymous class that matches a single-function interface by anonymously specifying the function:

```
new Runnable( () -> System.out.println("hello world"));
new Runnable( () - > {
  setSize(100, 100);
  setVisible(false); });
```

- This on-the-fly function definition in Java is also called a lambda expression

javax.swing.Timer

- code in the event dispatch thread should never sleep
- instead, such code can schedule an event for a later time
  - specified in milliseconds from now, e.g 500 is ½ a second

```
javax.swing.Timer timer =
  new javax.swing.Timer(500, myActionListener);
```

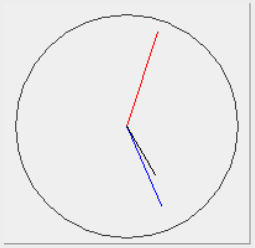
- `timer.start()` starts the timer
- this timer event is scheduled in the same way as user interaction events
- normally timers run forever

  `timer.setRepeat(false)` makes the timer run only once
- java.util.Timer is similar and useful for other things, but should not be used for displaying graphics

using a Timer for animation

```
public class Clock extends javax.swing.JFrame {
  public static final int offset = 32;
  private void drawHand(java.awt.Graphics g, int
clockRadius, int minutes, double lengthFraction)
{
    int centerX = clockRadius;
    int centerY = clockRadius;
    // in math 0 degrees is to the right
    // on the clock 0 minutes is vertical
    int mathAngle =
      (360 * minutes / 60 + 270) % 360;
    double angle = 2 * Math.PI *
                   (double)mathAngle / 360.0;
    double handLength =
      lengthFraction * (double)clockRadius;
    int endX = centerX +
      (int)Math.round(handLength *
                      Math.cos(angle));
    int endY = centerY +
      (int)Math.round(handLength *
                      Math.sin(angle));
    g.drawLine(centerX, centerY, endX, endY);
  }
```

```
  public Clock(int size) {
    java.awt.event.ActionListener updateClock =
      new java.awt.event.ActionListener() {
       public void actionPerformed(java.awt.event.ActionEvent e){
         repaint();
       }
      };
    new javax.swing.Timer(1000, updateClock).start();
    add(new javax.swing.JComponent() {
      protected void paintComponent(java.awt.Graphics g) {
        g.drawOval(10, 10, size - 20, size - 20);  // clock face
        java.util.GregorianCalendar cal =
          new java.util.GregorianCalendar();
        cal.setTime(new java.util.Date());
        drawHand(g, size / 2, cal.get(java.util.Calendar.SECOND),
                 0.85);
        drawHand(g, size / 2, cal.get(java.util.Calendar.MINUTE),
                 0.7);
        drawHand(g, size / 2, cal.get(java.util.Calendar.HOUR) * 5,
                 0.45);
      }
    });
    setSize(size, size + offset);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setVisible(true);
  }
  public static void main (String[] ignored) {
    Clock myClock = new Clock(200);
  }
```



AnimationTimer

- public abstract class AnimationTimer

  javafx.animation.AnimationTimer allows to create a timer, that is called in each frame while it is active. An extending class has to override the method `handle(long timestamp)` which will be called in every frame.

- frames are typically 60 times per second, or once every 16ms or so

Summary

- writing apps with swing
  - lots and lots of classes and methods to help write useful, reasonably-looking apps
  - layering: graphics “on top” (lower z) hide graphics “below”
- understand the thread model: all display actions must be done in the event handling thread
  - main thread can request actions to be executed in the event handling thread
  - timers can schedule actions to be executed in the event handling thread at a later time
- Android, iOS, have different systems but similar principles