Drawing and Java2D

15 February 2011
CMPT166
Sean Ho
Trinity Western University

What's on for today

- Menu bars and menus
- Scroll panes
- Window events and WindowAdapter class
- Drawing in Swing: paint / paintComponent()
  - Draw shapes
  - Fill, colours, and stroke
  - Text and images
  - Clipping
  - Coordinate transforms

Menus: JMenuBar

- JMenuBar: top-level container for menus

  ```
  JMenuBar bar = new JMenuBar();
  ```

  - Menubars contain menus and items
- Use the panel's existing layout manager:

  ```
  add( bar );
  ```
- Or fix it at the top of the window:

  ```
  setJMenuBar( bar );
  ```
- Can have multiple menubars per window

JMenu and JMenuItem

- A JMenu represents one menu (e.g., "File")

```
JMenu fileMenu = new JMenu();
bar.add( fileMenu );
```

- Contains menu items: JMenuItem

```
JMenuItem saveItem =
                      new JMenuItem( "Save" );
fileMenu.add( saveItem );
```

- Attach a handler to the menu item:

```
saveItem.addActionListener( handler );
```

- JMenu is itself a subclass of JMenuItem: this allows nested submenus

Scroll bars

- Widgets can be put inside scroll panes: show only a viewport of the whole widget
- e.g., a text area:

```
JTextArea blogEntry = new JTextArea(10, 40)
```

  - ➔ Only shows 10 lines, 40 characters of text

```
JScrollPane scrBlog = new JScrollPane(blogEntry);
```

  - ➔ Wrap in a scroll pane

```
add( scrBlog );
```

  - ➔ Add to a panel or window

- Scroll bar policy: whether to show

```
scrBlog.setVerticalScrollBarPolicy(
 JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED );
```

Window events

- We have seen: ActionEvent (button, menu)
  - also InputEvent (KeyEvent, MouseEvent)
- A WindowEvent is sent when the window interacts with the OS windowing system:
  - opening, closing, iconifying, activating
- A JFrame can register a window listener to handle these events:

```
myJFrame.setWindowListener( winevents );
```

- This handler must implement the WindowListener interface

Window listeners

- Implementing WindowListener means providing:

```
        class WinEvents implements WindowListener {
          public void windowOpened( WindowEvent e );
```

  - Also windowClosing, windowClosed, windowIconified, windowDeiconified, windowActivated, windowDeactivated
- Closing: once the close button is clicked
- Closed: after the window is done
- Activated: usually when click in window
  - Only one window may be active at a time

WindowAdapter class

- Implementing the WindowListener interface means needing to implement all its methods, even if you don't need them
- WindowAdapter is an abstract superclass that implements WindowListener and provides default blank bodies for the methods
- Subclass WindowAdapter and override just the ones you need:
  - class WinEvents extends WindowAdapter {

```
    public void windowClosed( WindowEvent e ) {
```

Swing graphics: .paint()

- JFrames have a .paint() method, which draws the window on the screen
  - To do our own drawing, override paint()
  - Make sure to call super.paint() first to draw the JFrame, then do our own drawing on top
- paint() takes a Graphics context as its argument
  - Drawing routines are methods of Graphics

```
public class SmileyFace extends JFrame {
    public void paint( Graphics g ) {
        super.paint( g );
        g.drawOval( .... );
```

paint() vs. paintComponent()

- JFrames: use paint() method
- JPanels and other JComponents: use paintComponent()
- paint() and paintComponent() are only called when a redraw is necessary
  - e.g., expose after being covered
- If you make a change and want to request a redraw, call
  - repaint() (method of JFrame or Jcomponents)
  - Actual repainting may happen a bit later

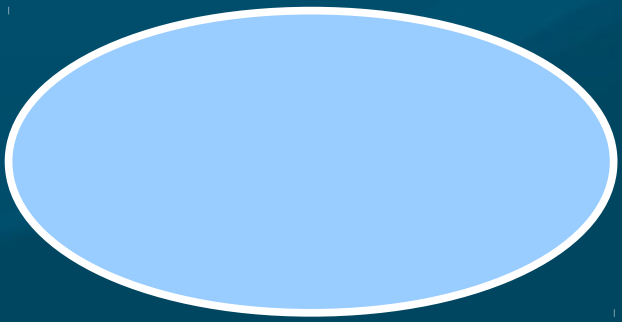
Lines and rectangles

- import java.awt.Graphics;

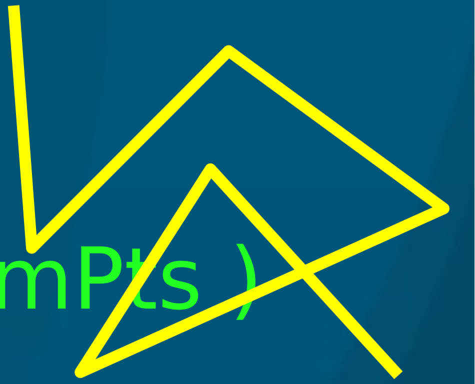
- g.drawLine(int x1, int y1, int x2, int y2);
  - Coordinates in pixels from top-left of component
- drawRect(x, y, w, h), fillRect
  - (x,y) is top-left corner of rectangle
- draw3DRect(x, y, w, h, boolean raised)
  - Border-shading so it looks raised or sunken
- drawRoundRect(x, y, w, h, arcW, arcH)
  - Specify diameter of rounded corners

Ovals and arcs

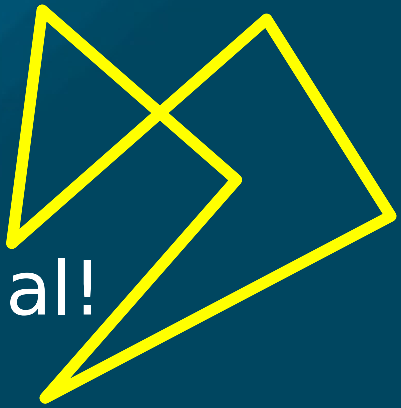
- g.drawOval(x, y, w, h), fillOval
  - Circles are ovals with equal width and height
- drawArc(x, y, w, h, angle, sweep), fillArc
  - Specify starting angle (0 points to right)
  - Specify how far the arc should go (sweep)
  - Angle and sweep are both in integer degrees

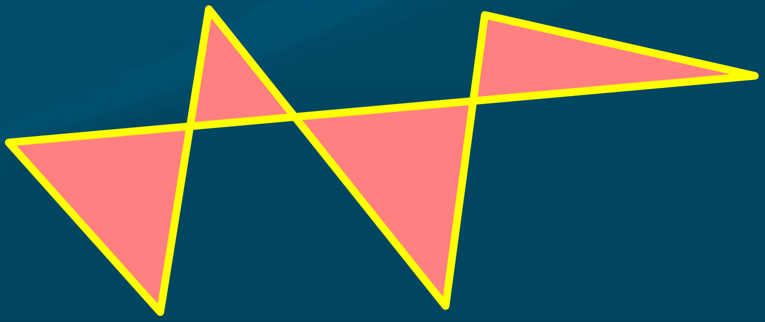


Polylines and polygons



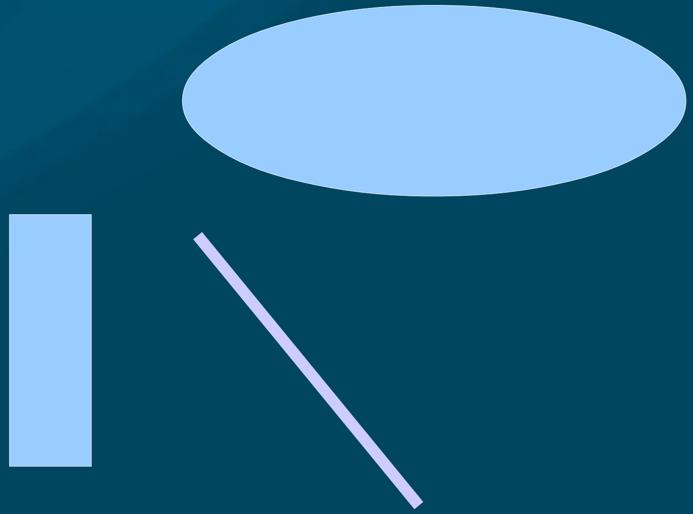
- drawPolyline(int[] x, int[] y, int numPts)
  - Arrays of x and y coordinates
  - Draws connected line segments
- drawPolygon(int[] x, int[] y, int numPts)
  - Connects last point to first point
- Also fillPolygon(...)
  - Filling an arbitrary polygon is not trivial! (tessellation)





Steps to draw in a widget

- Subclass JFrame and override paint()
  - Or JPanel and override paintComponent()
- Setup the current drawing context:
  - Pen colour, stroke, font,
    clip, coordinate system, etc.
- Basic drawing commands:
  - draw or fill:
  - Line, Rect, Oval, Arc



Colours (colors)

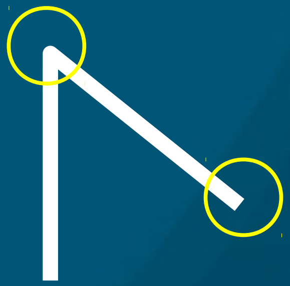
- **import java.awt.Color;**

- Set the current drawing colour before drawing the object:
  - **g.setColor(Color.BLUE);**
  - **g.drawArc(50, 50, 100, 100, 200, 140);**
  - **g.setColor(Color(0.7, 0.9, 0.1));**
  - **g.drawOval(80, 80, 40, 40);**
- A few named colours, or use an RGB triple
- JColorChooser: dialog to select a Color
  - **JColorChooser.showDialog(**
    this, “title”, defaultColor);

Line stroke

- Stroking is how lines are rendered
  - Line thickness:

```
g.setStroke( new BasicStroke(2f) );
```

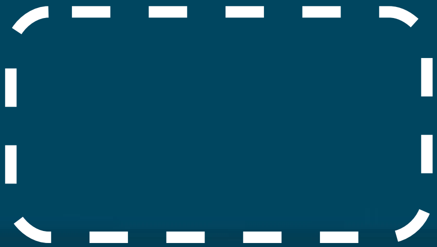


  - Cap style and join style:

```
new BasicStroke( 2f, BasicStroke.CAP_ROUND,
                     BasicStroke.JOIN_MITER, 10f )
```

  - Dash pattern and phase (offset):

```
float dash[] = { 10f, 5f };
new BasicStroke( 2f, ...ROUND, ...MITER, 10f,
                     dash, 0f )
```



Drawing text

- drawString(String text, int x, int y)
  - Uses current colour and font
- setFont(Font f)
  - Sets the current font in the graphics context
- Font class:

  Hello, World!

    - import java.awt.Font;
    - new Font(Font.SANS_SERIF, Font.PLAIN, 18)
  - Family (MONOSPACED, “Arial”, etc.)
  - Style: plain, italic, bold
  - Size: in points

Reading images from file

- ImageIO library knows jpg, gif, png, bmp

```
import javax.ImageIO;
```

- BufferedImage stores the image data:

```
BufferedImage img;
try {
    img = ImageIO.read(
                        new File( “apples.jpg” ) );
} catch (IOException e) { }
```

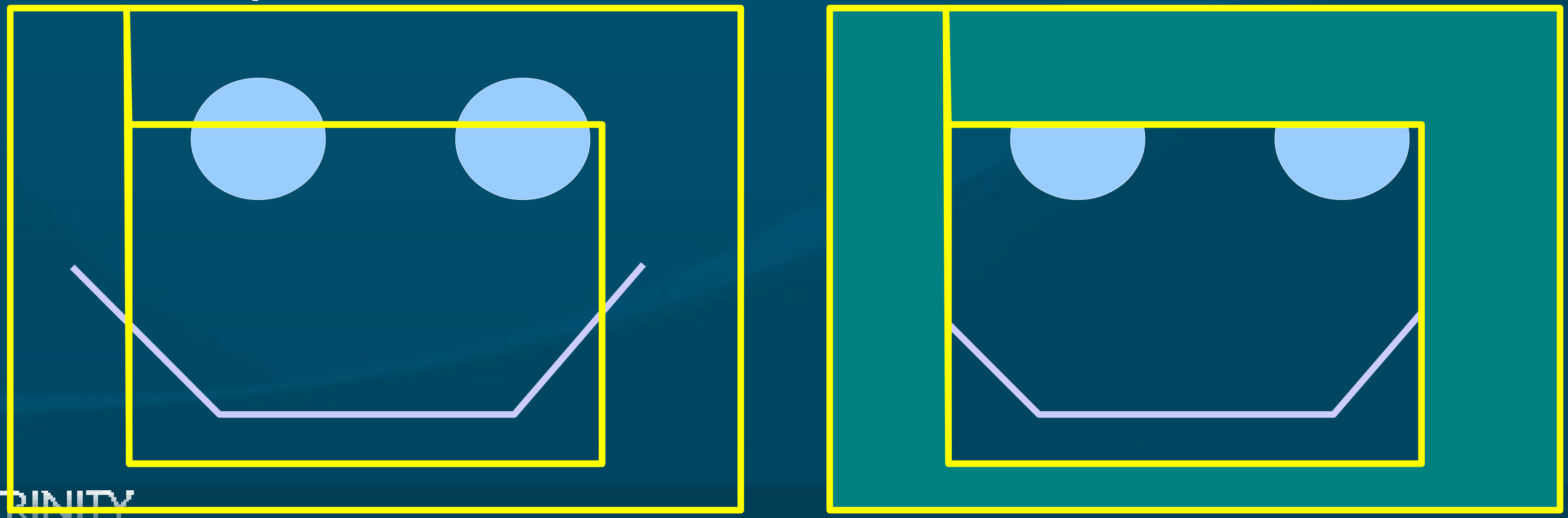
  - May raise IOException if file doesn't exist, etc

Drawing images on canvas

- g.drawImage(
  Image img, int x, int y, ImageObserver obs)
  - The ImageObserver is usually null
- Or select a sub-rectangle of the image and scale it to fit within a rectangle on canvas:
- g.drawImage(Image img, int
  dst_x$_1$, dst_y$_1$, dst_x$_2$, dst_y$_2$,
  src x$_1$, src y$_1$, src x$_2$, src y$_2$, ImgObs obs)
  - Source rectangle in the image
  - Destination rectangle in the canvas

Clipping

- The current clip is the viewport of the canvas which is being drawn on
  - Anything drawn outside the clip is not visible
  - Primitives (ovals, polygons, etc.) that lie partially outside the viewport are clipped to the viewport



Setting the clip region

- setClip(int x, y, w, h)
  - Sets the clip region to the given rectangle
  - Useful if you want to “protect” parts of the window/panel from being drawn over

- setClip() is also overloaded to take a Shape
  - For more complex clip regions
    - Polygon, Line2D, Arc2D, CubicCurve2D, etc.
  - See documentation for Shape interface

Coordinate transforms

- Default has origin at top-left, units in points: ~72 per inch
- An affine transform allows translation, rotation, scaling/flipping, and shearing
  - So you can draw in whatever coordinates you please
  - Convert from world coords to window coords
  - Each object can get its own coord system, too: object coords → world coords → window coords

Applying transforms

- Get a Graphics2D context:

```
g2 = (Graphics2D) g;
```

- Save old coordinate transform:

```
AffineTransform oldxf = g2.getTransform();
```

- Create and apply new transform:

```
AffineTransform xf = new AffineTransform();
xf.rotate( Math.toRadians(45) );
g2.transform( xf );
```

- After drawing, restore old transform:

```
g2.setTransform( oldxf );
```