cis3120
design and implementation of software applications I
spring 2015
lecture # II.3: using APIs and fundamental techniques in graphics

topics:

- Java Graphics API: applications, drawing basics

graphics - overview

- two classes used for rendering graphics are `java.awt.Graphics` and `java.awt.Graphics2D`
- `java.awt.Graphics` contains simple drawing primitives and functionality, extended by the `java.awt.Graphics2D` which supports more sophisticated functionality.
- in AWT, to change the look of a component override `paint(Graphics g)` method
- in Swing to change the look of a component override `paintComponent(Graphics g)` method (which calls `paint` when its ready)
- to create an empty canvas in Swing, a common method is to override `paintComponent` of a
  JPanel component
- `java.awt.Graphics` contains drawing primitives: lines, Strings, rectangles, ovals, arcs. in addition to polygon, polyline convenience methods.

`Graphics` - line and string

- simple methods from the `java.awt.Graphics` class
- `void drawLine( int x1, int y1, int x2, int y2 );`
  - draws a line connecting (x1,y1) and (x2,y2);
- `void drawString( String str, int x, int y );`
  - draws the text in "str", with its lower left corner at (x,y)

`Graphics` - line and string example

```
public class DrawString {
  public static void main(String[] args) {
    JFrame frame = new JFrame();
    frame.setSize(500,400);
    frame.setTitle("Draw String");
    frame.getContentPane().setBackground(Color.black);

    // add an instance of an anonymous inner class that extends JPanel
    frame.getContentPane().add(new JPanel(){
        public void paintComponent(Graphics g) {
          setOpaque(false);
          g.setColor(Color.green);
          g.drawString("Hello world!", 100, 100);
          g.drawLine(100,125,400,125);
        }
      });

    frame.setVisible(true);
  }
}
```

Graphics - rectangle, oval, arc (1)

- bounding rectangles
  - used for drawing rectangular shapes as well as ovals by defining their bounding rectangles
  - coordinates of origin (upper left corner)
  - extent (width and height)
- arcs
  - measured in degrees
  - starting from 0° (along positive X-axis)
  - extent (total angle of arc)

Graphics - rectangle, oval, arc (2)

- methods from the `java.awt.Graphics` class for drawing outlines of shapes
- `void drawRect( int x, int y, int width, int height );`
  - draws a rectangle with its upper left corner at (x,y), extending the specified "width" and "height"
- `void drawOval( int x, int y, int width, int height );`
  - draws an oval circumscribed in the bounding rectangle with its upper left corner at (x,y), extending the specified "width" and "height"
- `void drawArc( int x, int y, int width, int height, int startAngle, int arcAngle );`
  - draws an arc whose oval is circumscribed in the bounding rectangle with its upper left corner at (x,y), extending the specified "width" and "height", where the arc starts at the "startAngle", measured in degrees (where 0°) is horizontal along the positive x-axis), extending for "arcAngle" degrees

Graphics - rectangle, oval, arc (3)

- `void setColor( Color color );`
  - sets the foreground (pen) color to the specified color
- methods from the `java.awt.Graphics` class for drawing filled shapes:
- `void fillRect( int x, int y, int width, int height );`
  - draws a filled rectangle with its upper left corner at (x,y), extending the specified "width" and "height"
- `void fillOval( int x, int y, int width, int height );`
  - draws a filled oval circumscribed in the bounding rectangle with its upper left corner at (x,y), extending the specified "width" and "height"
- `void fillArc( int x, int y, int width, int height, int startAngle, int arcAngle );`
  - draws a filled arc whose oval is circumscribed in the bounding rectangle with its upper left corner at (x,y), extending the specified "width" and "height", where the arc starts at the "startAngle", measured in degrees (where 0°) is horizontal along the positive x-axis), extending for "arcAngle" degrees

Graphics - rectangle, oval, arc example

```
public class DrawRect {
  public static void main(String[] args) {
    JFrame frame = new JFrame();
    ... // set size, title and background color
    frame.getContentPane().add(new JPanel(){
        public void paintComponent(Graphics g) {
          setOpaque(true);
          setBackground(Color.black);
          g.setColor(Color.green);
          g.drawRect(50,50,200,50);
          g.fillRect(300,50,200,50);
          g.setColor(Color.red);
          g.drawOval(50,150,200,50);
          g.fillOval(300,150,200,50);
          g.setColor(Color.blue);
          g.drawArc(50,250,200,200,25,200);
          g.fillArc(300,250,200,200,0,45);
        }
      });
    frame.setVisible(true);
  }}
```

Graphics - polygon, polyline

- methods from the `java.awt.Graphics` class for drawing polygons
- `void drawPolygon(int[] xPoints, int[] yPoints, int nPoints);`
  - draws a closed polygon defined by arrays of x and y coordinates
- `void drawPolygon( Polygon p );`
  - draws the outline of a polygon defined by the specified `Polygon` object
- `void drawPolyline( int[] xPoints, int[] yPoints, int nPoints );`
  - draws a sequence of connected lines defined by arrays of x and y coordinates
- the first two have counterparts for drawing filled polygons:
  - `void fillPolygon(int[] xPoints, int[] yPoints, int nPoints);`
  - `void fillPolygon( Polygon p );`

Graphics - polygon, polyline example

```
public class DrawPoly {
  public static void main(String[] args) {
    JFrame frame = new JFrame();
    ... // set size, title and background color
    frame.getContentPane().add(new JPanel(){
        public void paintComponent(Graphics g) {
          setOpaque(false);
          g.setColor(Color.green);
          int[] xs = new int[]{100, 200, 300, 400, 250};
          int[] ys = new int[]{100, 90, 60, 125, 200};
          g.drawPolygon(xs, ys, xs.length);
          g.setColor(Color.red);
          int[] xs2 = new int[]{100, 200, 250, 300, 400, 250};
          int[] ys2 = new int[]{350, 350, 250, 350, 350, 450};
          g.fillPolygon(xs2, ys2, xs2.length);
        }
      });
    frame.setVisible(true);
  }}
```


custom color

- `java.awt.Color` class
- color is defined using the "RGB" methodology
- "Red", "Green", "Blue"
- each is an integer between 0 and 255, where 0 means no color and 255 means maximum color
- so white is: red=255 green=255 blue=255 or the ordered triple (255,255,255)
  - and black is: red=0 green=0 blue=0
  - and red is: red=255 green=0 blue=0
  - and green is: red=0 green=255 blue=0
  - and blue is: red=0 green=0 blue=255
- make up your own colors...

Graphics2D class

- `Graphics2D` class contains more powerful set of operations than `Graphics` (i.e. geometric transformations, modifying line properties etc.)
- both the `paint` and `paintComponent` methods receive a `Graphics2D` object, but for compatibility reasons, the parameter type for these methods remain `Graphics`
- in order to use the `Graphics2D` methods, you have to cast the passed object reference in `paintComponent`:

```
public void paintComponent(Graphics g) {
  Graphics2D g2 = (Graphics2D) g;
  ...
}
```

- `Graphics2D` provides methods for drawing three kinds of objects: shapes (`draw()` and `fill()`), text (`drawString()`) and images (`drawImage()`).

Graphics2D - shapes

- Shape class is the parent of all the shapes in Graphics2D. Subclasses of the Shape are contained in the package `java.awt.geom`
- some useful subclasses of Shape
  - Line2D
  - Rectangle2D
  - Ellipse2D
- all above are abstract but their concrete implementation can be accessed by specifying the type .Float or .Double:

```
Shape r = Rectangle2D.Float(x, y, width, height);
```

- once a shape object is created it can be drawn using the `draw()` method:

```
draw(r);
```


Graphics2D - shapes example

```
import java.awt.*;
import javax.swing.*;
import java.awt.geom.*;

public class DrawShape2D {
  public static void main(String[] args) {
    JFrame frame = new JFrame();

    frame.setSize(500,500);
    frame.setTitle("Draw Line2D, Rectangle2D, Ellipse 2D");
    frame.getContentPane().setBackground(Color.black);

    frame.getContentPane().add(new JPanel(){
        public void paintComponent(Graphics g) {
          Graphics2D g2 = (Graphics2D) g;
          setOpaque(false);

          g2.setPaint(Color.red);
          Line2D line = new Line2D.Float(50,50,300,50);
          g2.setStroke(new BasicStroke(4));
          g2.draw(line);
```



```
          g2.setPaint(Color.blue);
          g2.setStroke(new BasicStroke(1));
          Rectangle2D rect1 = new Rectangle2D.Float(100,100,100,200);
          Rectangle2D rect2 = new Rectangle2D.Float(300,100,100,200);
          g2.draw(rect1);
          g2.fill(rect2);

          g2.setPaint(Color.green);
          g2.setStroke(new BasicStroke(2));
          Ellipse2D oval1 = new Ellipse2D.Float(100,100,100,200);
          Ellipse2D oval2 = new Ellipse2D.Float(100,350,200,100);
          g2.draw(oval1);
          g2.fill(oval2);
        }
      });

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }
}
```


fonts (1)

- fonts in Java are defined using the `java.awt.Font` class
- you can see which fonts (by name) are available in your system by using the `java.awt.GraphicsEnvironment.getAllFonts()` method
- you can get information about the point size of the font, whether it is italic, bold or plain
- you will most likely want to get the size of a string that might be drawn with the current font. first you need to create a `FontMetrics` object, then you can call the `FontMetrics.stringWidth( String str )` method to find the width

fonts (2)

- useful font properties available in `FontMetrics`:
  - **ascent** — the distance from the font's baseline to the top of an alphanumeric character
    use `int FontMetrics.getMaxAscent()`
  - **descent** — the distance from the font's baseline to the bottom of an alphanumeric character with descenders use `int FontMetrics.getMaxDescent()`
  - **height** — the distance between the baseline of adjacent lines of text; the sum of the leading + ascent + descent.
  - **leading** — aka interline spacing; the logical amount of space to be reserved between the descent of one line of text and the ascent of the next line
  - **advance** — the distance from the leftmost point to the rightmost point on the string's baseline use `int FontMetrics.charWidth( char ch )` to get the advance of the char argument

fonts example

```
public class DrawString2D {
  public static void main(String[] args) {
    JFrame frame = new JFrame();
    ... // set size, title and background color
    frame.getContentPane().add(new JPanel(){
        public void paintComponent(Graphics g) {
          Graphics2D g2 = (Graphics2D) g;
          setOpaque(false);
          g2.setPaint(Color.GREEN);
          Font sansbold14 = new Font("SansSerif", Font.BOLD, 14);
          g2.setFont(sansbold14);
          g2.drawString("Hello world!", 100, 100);
          Font arialitalic16 = new Font("Arial", Font.ITALIC, 16);
          g2.setFont(arialitalic16);
          g2.drawString("Hello world!", 100, 250);
        }
      });
    frame.setVisible(true);
  }
}
```


images

- images are not painted like the the shapes or text.
- images are loaded from files and drawn
- load them using `java.awt.imageio.read(File)`:

  `Image image = ImageIO.read(new File("some.jpg"));`

  this method throws `IOException` if the file is not found
- draw them using `Graphics.drawImage()`:

  `g.drawImage(image, x, y, ImageObserver);`
- note that an `Image` is a Java object unto itself, defined in the `java.awt` package

images example

```
import java.awt.*;
import java.io.*;
import javax.swing.*;
import javax.imageio.*;

public class ImageGui {
  private BufferedImage myImage;

  public static void main(String[] args) {
    ImageGui gui = new ImageGui();
    JFrame frame = new JFrame();
    frame.getContentPane().setLayout(new BorderLayout(5,5));
    frame.getContentPane().setBackground(Color.white);
    Label lab = new Label("this is an imported image");
    frame.getContentPane().add(lab, BorderLayout.NORTH);
    frame.setSize(670,570);
    frame.setTitle("Mets game 15 Sept. 2007");

    try {
        gui.myImage = ImageIO.read(new File("mets-game-15sep2007.jpg"));
    }
```

```
    catch(IOException exp) {
        System.out.println("Could not read the image file!");
    }

    ImageCanvas myCanvas = new ImageCanvas(gui.myImage);
    frame.getContentPane().add(myCanvas, BorderLayout.CENTER);
    frame.setVisible(true);
  }
}

class ImageCanvas extends JPanel {
  Image myImage;

  ImageCanvas(Image myImage){
    this.myImage = myImage;
    setSize(640,480);
  }

  public void paintComponent(Graphics g) {
    g.drawImage(myImage, 10, 10, null);
  }
}
```