Lecture 2J - Graphics and Java2D

Outline

11.1 Introduction

- In this chapter
  - Draw 2D shapes
  - Colors
  - Fonts
- Java appealing for its graphics support
  - Has a class hierarchy for its graphics classes and 2D API classes
- Java coordinate system
  - (x,y) pairs
    - x - horizontal axis
    - y - vertical axis
  - Upper left corner is (0,0)
  - Coordinates measured in pixels (smallest unit of resolution)

11.2 Graphics Contexts and Graphics Objects

- Graphics context
  - Enables drawing on screen
  - **`Graphics`** object manages graphics context
    - Controls how information is drawn
    - Has methods for drawing, font manipulation, etc
    - We have used **`Graphics`** object **`g`** for applets
  - **`Graphics`** an **`abstract`** class
    - Cannot instantiate objects
    - Implementation hidden - more portable
- Class **`Component`**
  - Superclass for many classes in **`java.awt`**
  - Method **`paint`** takes **`Graphics`** object as argument

11.2 Graphics Contexts and Graphics Objects

- Class **`Component`** (continued)
  - **`paint`** called automatically when applet starts
  - **`paint`** often called in response to an event
  - **`repaint`** calls **`update`**, which forces a **`paint`** operation
    - **`update`** rarely called directly
    - Sometimes overridden to reduce "flicker"

    Headers:

    ```
    public void repaint()
    public void update( Graphics g )
    ```
- In this chapter
  - Focus on **`paint`** method

11.3 Color Control

- Class **`Color`**
  - Defines methods and constants for manipulating colors
  - Colors created from red, green, and blue component
    - RGB value: 3 integers from 0 to 255 each, or three floating point values from 0 to 1.0 each
    - Larger the value, more of that color
  - Color methods **`getRed`**, **`getGreen`**, **`getBlue`** return an integer between 0 and 255 representing amount

```
19       g.setColor( new Color( 255, 0, 0 ) );
```

  - Graphics method **`setColor`** sets drawing color
    - Takes **`Color`** object
  - Method **`getColor`** gets current color setting
  - There are color constants
    - **`public final static Color yellow; ...`**

```
1  // Fig. 11.5: ShowColors.java
2  // Demonstrating Colors
3  import java.awt.*;
4  import javax.swing.*;
5  import java.awt.event.*;
6
7  public class ShowColors extends JFrame {
8     public ShowColors()
9     {
10       super( "Using colors" );
11
12       setSize( 400, 130 );
13       show();
14    }
15
16    public void paint( Graphics g )
17    {
18       // set new drawing color using integers
19       g.setColor( new Color( 255, 0, 0 ) );
20       g.fillRect( 25, 25, 100, 20 );
21       g.drawString( "Current RGB: " + g.getColor(), 130, 40 );
22
23       // set new drawing color using floats
24       g.setColor( new Color( 0.0f, 1.0f, 0.0f ) );
25       g.fillRect( 25, 50, 100, 20 );
26       g.drawString( "Current RGB: " + g.getColor(), 130, 65 );
27
```

Outline

1. `import`

1.1 Class definition

1.2 Define `paint`

Call to **`setColor`**, takes a **`Color`** object.

`fillRect` fills a rectangle (more later)

```
28       // set new drawing color using static Color objects
29       g.setColor( Color.blue );
30       g.fillRect( 25, 75, 100, 20 );
31       g.drawString( "Current RGB: " + g.getColor(), 130, 90 );
32
33       // display individual RGB values
34       Color c = Color.magenta;
35       g.setColor( c );
36       g.fillRect( 25, 100, 100, 20 );
37       g.drawString( "RGB values: " + c.getRed() + ", " +
38          c.getGreen() + ", " + c.getBlue(), 130, 115 );
39    }
40
41    public static void main( String args[] )
42    {
43       ShowColors app = new ShowColors();
44
45       app.addWindowListener(
46          new WindowAdapter() {
47             public void windowClosing( WindowEvent e )
48             {
49                System.exit( 0 );
50             }
51          }
52       );
53    }
54 }
```

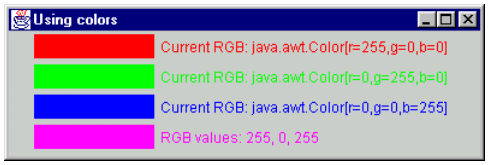
Outline

1.2 Define `paint`

1.3 `main`

Outline

Program Output



11.4 Font Control

- Class **Font**
  - Constructor takes three arguments

```
public Font( String name, int style, int size)
```

    - name: any font supported by system (**Serif**, **Monospaced**)
    - style: constants **FONT.PLAIN**, **FONT.ITALIC**, **FONT.BOLD**
      - Combinations: **FONT.ITALIC + FONT.BOLD**
    - size: measured in points (1/72 of an inch)

  - Use similar to **Color**
  - **g.setFont(fontObject);**

```
20       g.setFont( new Font( "Serif", Font.BOLD, 12 ) );
```


11.4 Font Control

- Methods
  - **getStyle()**
  - **getSize()**
  - **getName()**
  - **getFamily()**
  - **isPlain()**
  - **isBold()**
  - **isItalic()**
  - **getFont()**
  - **setFont(Font f)**

```
1  // Fig. 11.9: Fonts.java
2  // Using fonts
3  import java.awt.*;
4  import javax.swing.*;
5  import java.awt.event.*;
6
7  public class Fonts extends JFrame {
8     public Fonts()
9     {
10       super( "Using fonts" );
11
12       setSize( 420, 125 );
13       show();
14    }
15
16    public void paint( Graphics g )
17    {
18       // set current font to Serif (Times), bold, 12pt
19       // and draw a string
20       g.setFont( new Font( "Serif", Font.BOLD, 12 ) );
21       g.drawString( "Serif 12 point bold.", 20, 50 );
22
23       // set current font to Monospaced (Courier),
24       // italic, 24pt and draw a string
25       g.setFont( new Font( "Monospaced", Font.ITALIC, 24 ) );
26       g.drawString( "Monospaced 24 point italic.", 20, 70 );
27
```

Notice use of **`Font`** objects and method calls

Outline

Font Example

1. `import`

1.1 Constructor

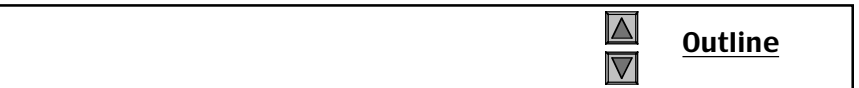
2. `paint`

```
28       // set current font to SansSerif (Helvetica),
29       // plain, 14pt and draw a string
30       g.setFont( new Font( "SansSerif", Font.PLAIN, 14 ) );
31       g.drawString( "SansSerif 14 point plain.", 20, 90 );
32
33       // set current font to Serif (times), bold/italic,
34       // 18pt and draw a string
35       g.setColor( Color.red );
36       g.setFont(
37          new Font( "Serif", Font.BOLD + Font.ITALIC, 18 ) );
38       g.drawString( g.getFont().getName() + " " +
39                     g.getFont().getSize() +
40                     " point bold italic.", 20, 110 );
41    }
42
43    public static void main( String args[] )
44    {
45       Fonts app = new Fonts();
46
47       app.addWindowListener(
48          new WindowAdapter() {
49             public void windowClosing( WindowEvent e )
50             {
51                System.exit( 0 );
52             }
53          }
54       );
55    }
56 }
```

Outline

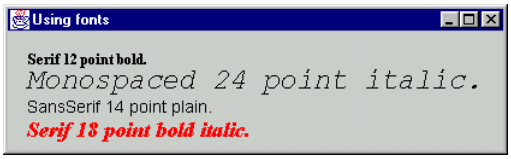
2. `paint`

3. `main`



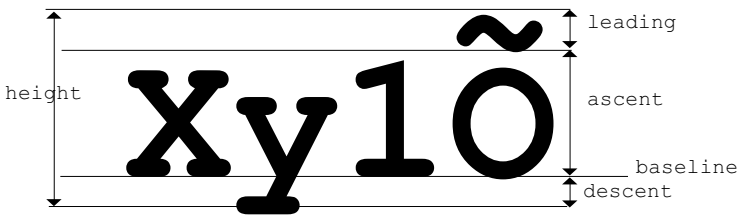
Outline

Program Output



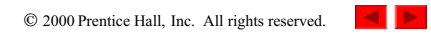
11.4 Font Control

- Class **`FontMetrics`**
  - Has methods for getting font metrics
  - **`g.getFontMetrics`** - returns **`FontMetrics`** object for current font



11.4 Font Control

- **`FontMetrics`** (and **`Graphics`**) methods
  - **`getAscent()`**
  - **`getDescent()`**
  - **`getLeading()`**
  - **`getHeight()`**
  - **`getFontMetrics()`**
  - **`getFontMetrics( Font f )`**



```
1  // Fig. 11.12: Metrics.java
2  // Demonstrating methods of class FontMetrics and
3  // class Graphics useful for obtaining font metrics
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Metrics extends JFrame {
9     public Metrics()
10    {
11       super( "Demonstrating FontMetrics" );
12
13       setSize( 510, 210 );
14       show();
15    }
16
17    public void paint( Graphics g )
18    {
19       g.setFont( new Font( "SansSerif", Font.BOLD, 12 ) );
20       FontMetrics fm = g.getFontMetrics();
21       g.drawString( "Current font: " + g.getFont(), 10, 40 );
22       g.drawString( "Ascent: " + fm.getAscent(), 10, 55 );
23       g.drawString( "Descent: " + fm.getDescent(), 10, 70 );
24       g.drawString( "Height: " + fm.getHeight(), 10, 85 );
25       g.drawString( "Leading: " + fm.getLeading(), 10, 100 );
26
```

Create new **`fontMetrics`** object, with fontmetrics for current font.

Outline

FontMetrics Example

1. `import`

1.1 Constructor

1.2 `paint`

```
27       Font font = new Font( "Serif", Font.ITALIC, 14 );
28       fm = g.getFontMetrics( font );
29       g.setFont( font );
30       g.drawString( "Current font: " + font, 10, 130 );
31       g.drawString( "Ascent: " + fm.getAscent(), 10, 145 );
32       g.drawString( "Descent: " + fm.getDescent(), 10, 160 );
33       g.drawString( "Height: " + fm.getHeight(), 10, 175 );
34       g.drawString( "Leading: " + fm.getLeading(), 10, 190 );
35    }
36
37    public static void main( String args[] )
38    {
39       Metrics app = new Metrics();
40
41       app.addWindowListener(
42          new WindowAdapter() {
43             public void windowClosing( WindowEvent e )
44             {
45                System.exit( 0 );
46             }
47          }
48       );
49    }
50 }
```

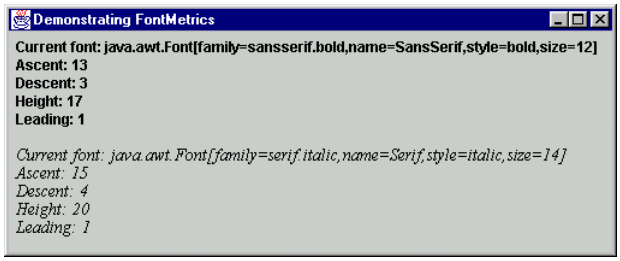
Outline

1.2 `paint`

1.3 `main`

Outline

Program Output



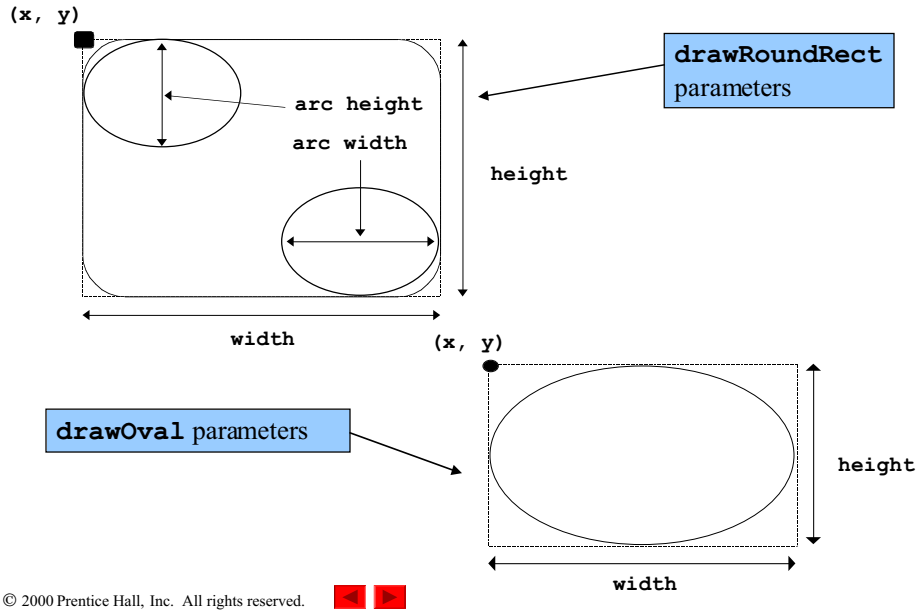
11.5 Drawing Lines, Rectangles and Ovals

- Graphics methods for drawing shapes
  - **`drawLine( x1, y1, x2, y2 )`**
    - Line from **`x1`**, **`y1`** to **`x2`**, **`y2`**
  - **`drawRect( x1, y1, width, height)`**
    - Draws rectangle with upper left corner **`x1, y1`**
  - **`fillRect( x1, y1, width, height)`**
    - As above, except fills rectangle with current color
  - **`clearRect (x1, y1, width, height)`**
    - As above, except fills rectangle with background color
  - **`draw3DRect(x1, y1, width, height, isRaised)`**
    - Draws **`3D`** rectangle, raised if **`isRaised true,`** else lowered

11.5 Drawing Lines, Rectangles and Ovals

- Graphics methods for drawing shapes (continued)
  - **`fill3DRect`**
    - As previous, but fills rectangle with current color
  - **`drawRoundRect( x, y, width, height, arcWidth, arcHeight )`**
    - Draws rectangle with rounded corners. See diagram next slide.
  - **`fillRoundRect( x, y, width, height, arcWidth, arcHeight )`**
  - **`drawOval( x, y, width, height )`**
    - Draws oval in bounding rectangle (see diagram)
    - Touches rectangle at midpoint of each side
  - **`fillOval ( x, y, width, height )`**

11.5 Drawing Lines, Rectangles and Ovals




```
1  // Fig. 11.14: LinesRectsOvals.java
2  // Drawing lines, rectangles and ovals
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class LinesRectsOvals extends JFrame {
8     private String s = "Using drawString!";
9
10    public LinesRectsOvals()
11    {
12       super( "Drawing lines, rectangles and ovals" );
13
14       setSize( 400, 165 );
15       show();
16    }
17
18    public void paint( Graphics g )
19    {
20       g.setColor( Color.red );
21       g.drawLine( 5, 30, 350, 30 );
22
23       g.setColor( Color.blue );
24       g.drawRect( 5, 40, 90, 55 );
25       g.fillRect( 100, 40, 90, 55 );
26
27       g.setColor( Color.cyan );
28       g.fillRoundRect( 195, 40, 90, 55, 50, 50 );
29       g.drawRoundRect( 290, 40, 90, 55, 20, 20 );
30
```

Draw various shapes using **Graphics** methods.

Outline

Lines, Rectangles, and Ovals Example

1. `import`

1.1 Constructor

2. `paint`

```
31       g.setColor( Color.yellow );
32       g.draw3DRect( 5, 100, 90, 55, true );
33       g.fill3DRect( 100, 100, 90, 55, false );
34
35       g.setColor( Color.magenta );
36       g.drawOval( 195, 100, 90, 55 );
37       g.fillOval( 290, 100, 90, 55 );
38    }
39
40    public static void main( String args[] )
41    {
42       LinesRectsOvals app = new LinesRectsOvals();
43
44       app.addWindowListener(
45          new WindowAdapter() {
46             public void windowClosing( WindowEvent e )
47             {
48                System.exit( 0 );
49             }
50          }
51       );
52    }
53 }
```

Outline

2. `paint`

3. `main`

Outline

Program Output