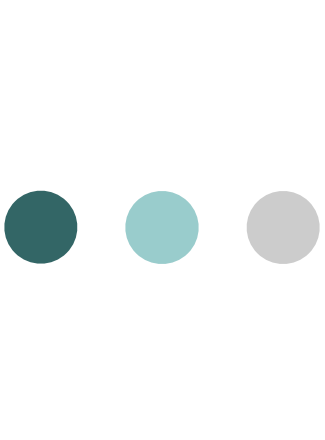


GUI 2D

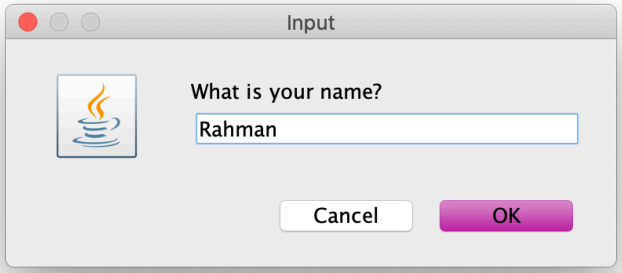
Using Components to create GUI

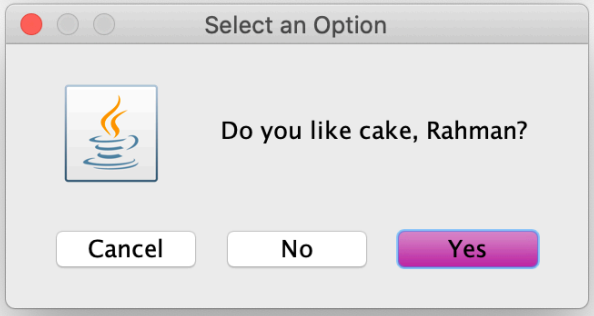
JOptionPane Dialog

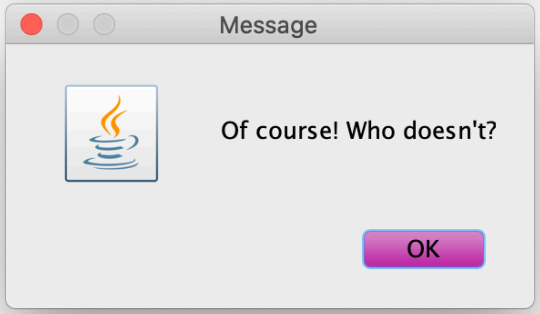
- JOptionPane can be used in three major ways: to display a message (as just demonstrated), to present a list of choices to the user, and to ask the user to type input.
- The three methods that implement these three behaviors are called showMessageDialog, showConfirmDialog, and showInputDialog, respectively.

JOptionPane Dialog Example

```
// read the user's name graphically
String name =
JOptionPane.showInputDialog(null, "What is
your name?");

// ask the user a yes/no question
int choice =
JOptionPane.showConfirmDialog(null,"Do you
like cake, " + name + "?");

// show different response depending on answer
if (choice == JOptionPane.YES_OPTION) {
   JOptionPane.showMessageDialog(null, "Of
   course! Who doesn't?");
}
else { // choice == NO_OPTION or CANCEL_OPTION

   JOptionPane.showMessageDialog(null, "We'll
   have to agree to disagree."); }
}
```

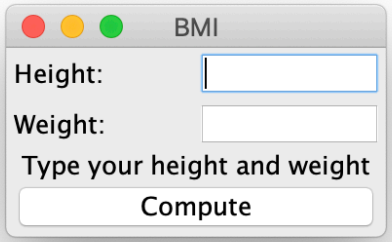






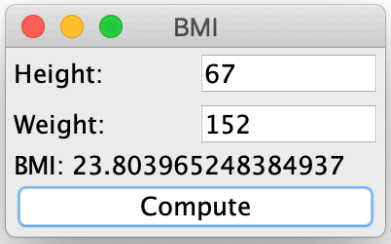
Multiple GUI Events Example

```
public BMIEvents(){
        // set up components
        heightField = new JTextField(5);
        heightField.addActionListener(this);
        weightField = new JTextField(5);
        weightField.addActionListener(this);
        bmiLabel = new JLabel( "  Type your height and weight  ");
        computeButton = new JButton("Compute");
        computeButton.addActionListener(this);

        // layout
        JPanel north = new JPanel(new GridLayout(2, 2));
        north.add(new JLabel(" Height: "));
        north.add(heightField);
        north.add(new JLabel(" Weight: "));
        north.add(weightField);

        // overall frame
        frame = new JFrame("BMI");

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new BorderLayout());
        frame.add(north, BorderLayout.NORTH);
        frame.add(bmiLabel, BorderLayout.CENTER);
        frame.add(computeButton, BorderLayout.SOUTH);
        frame.pack();
        frame.setVisible(true);
}
```



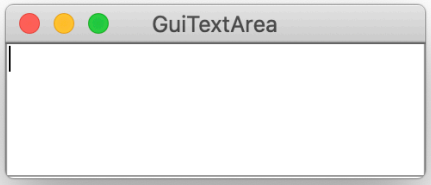
Multiple GUI Events Example

```
// Handles clicks on Compute button by computing the BMI.
public void actionPerformed(ActionEvent event) {
KeyboardFocusManager keyManager = KeyboardFocusManager.getCurrentKeyboardFocusManager();

if(event.getSource().equals(computeButton))
{         // read height and weight info from text fields
          double height = Double.parseDouble(heightField.getText());
          double weight = Double.parseDouble(weightField.getText());
          // compute BMI and display it onscreen
          double bmi = weight / (height * height) * 703;
          bmiLabel.setText(" BMI: " + bmi);
          // move focus back to heightField
          keyManager.focusNextComponent();
}
// enter event of the heightField text object
else if (event.getSource().equals(heightField)){
          try{
                    Double.parseDouble(heightField.getText());
                    // move focus to weightField
                    keyManager.focusNextComponent();
          }
          catch(Exception e){
                    heightField.setText(""); // error: clear text
          }

}
else if (event.getSource().equals(weightField)){
          try{
                    Double.parseDouble(weightField.getText());
                    // move focus back to computeButton
                    keyManager.focusNextComponent();
          }
          catch(Exception e){
                    weightField.setText(""); // error: clear text
          }
}
}
```



GUI Text Areas, Scrollbars, and Fonts

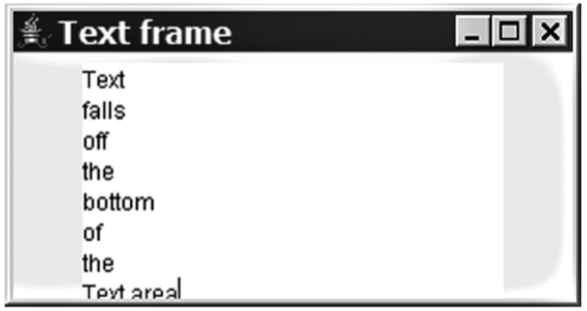
```
JTextArea linesOfText;
public GuiTextArea(){

      linesOfText = new JTextArea(5, 20);
      // overall frame
      frame = new JFrame("GuiTextArea");
      frame.setLayout(new GridLayout(2, 2));

      frame.add(linesOfText);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.pack();
      frame.setVisible(true);

}
```



GUI Text Areas, Scrollbars, and Fonts

- In a larger GUI example, an event listener might examine the text that is written in a text area by calling its getText method or set new text in the text area by calling its setText method.

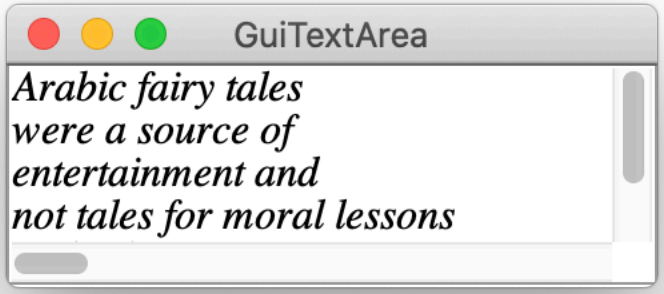
- Currently, when the user types too much text to fit in the text area, the text disappears off the bottom of the text box:



GUI Text Areas, Scrollbars, and Fonts

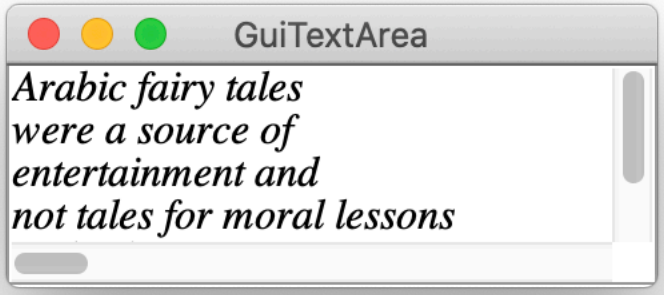
- To fix this problem, we can make the text area scrollable by adding familiar navigation components called scrollbars to it.
- Scrollbars are represented by instances of a special container component called a JScrollPane.
- To make a component scrollable, create a JScrollPane, add the component to the scroll pane, and then add the scroll pane to the overall frame.
- You construct a JScrollPane object by passing the relevant component as a parameter:

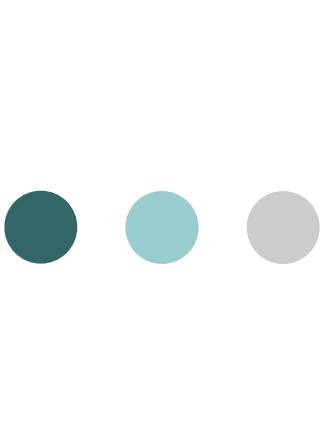
- `// use scrollbars on this text area`
  `frame.add(new JScrollPane(area));`



GUI Text Areas, Scrollbars, and Fonts

```
linesOfText = new JTextArea(5, 20);
linesOfText.setFont(new Font("Serif", Font.ITALIC, 16));
// overall frame
frame = new JFrame("GuiTextArea");
frame.setLayout(new GridLayout(1, 1));


frame.add(new JScrollPane(linesOfText));
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.pack();
frame.setVisible(true);
```





Graphics 2D

Painting in Java GUI

Custom components

- AWT/Swing come with lots of components that you can use to implement a fully-featured GUI.
- But there are cases when you need a custom component.
  - Usually this is when you want to paint custom 2-D graphics.
  - We often call a custom painted component a canvas.
- To do so, write a class that extends JComponent .
  - Override method paintComponent to tell Java how to draw it:
  - public void paintComponent(Graphics g)
    - Some programmers extend JPanel rather than JComponent .

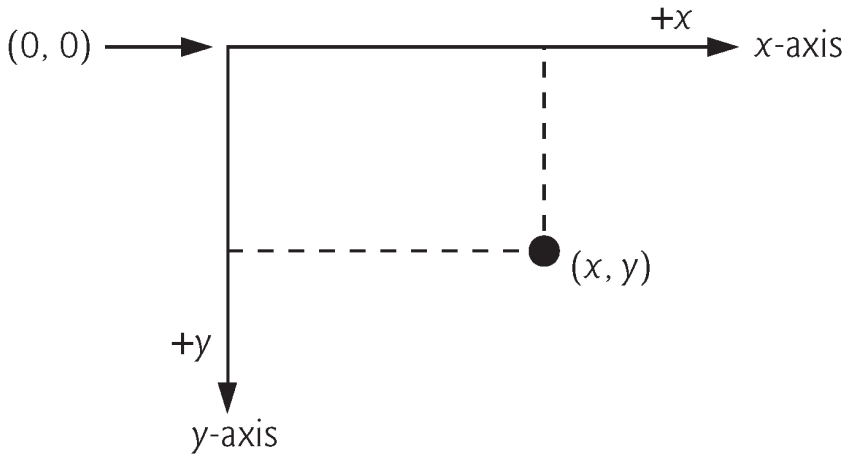
Quick drawing example

```
public class MyCanvas extends JComponent {
    public MyCanvas() {
        this.setBackground(Color.WHITE);
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g2.setPaint(Color.BLUE);
        g2.fillOval(10, 10, 20, 50);
    }
}
```

Coordinate System

- Coordinate system
  - a scheme for identifying every point on the screen.
- The upper-left corner of a GUI component (e.g., a window) has the coordinates (0, 0).
- A coordinate pair is composed of an x-coordinate (the horizontal coordinate) and a y-coordinate (the vertical coordinate).
  - x-coordinates from left to right.
  - y-coordinates from top to bottom.
- The x-axis describes every horizontal coordinate, and the y-axis every vertical coordinate.
- Coordinate units are measured in pixels.
  - A pixel is a display monitor's smallest unit of resolution.

Coordinate System



Coordinate System

- Upside-down Cartesian

- Ywindow = height - Ycartesian

Graphics Context/Object

- A graphics context enables drawing on the screen.
- A Graphics object manages a graphics context and draws pixels on the screen.
- Graphics objects contain methods for drawing, font manipulation, color manipulation and the like.
- Class JComponent (package javax.swing) contains a paintComponent for drawing graphics.
  - Takes a Graphics object as an argument.
  - Passed to the paintComponent method by the system when a lightweight Swing component needs to be repainted.

Graphics Context/Object

- When you create a GUI-based application, one of those threads is known as the event-dispatch thread (EDT) and it is used to process all GUI events.
- All drawing and manipulation of GUI components should be performed in that thread.
- The application container calls method paintComponent (in the EDT) for each lightweight component as the GUI is displayed.
- If you need paintComponent to execute, you can call method repaint, which is inherited by all JComponents indirectly from class Component (package java.awt).

Class Color

Color constant	RGB value
`public final static Color RED`	255, 0, 0
`public final static Color GREEN`	0, 255, 0
`public final static Color BLUE`	0, 0, 255
`public final static Color ORANGE`	255, 200, 0
`public final static Color PINK`	255, 175, 175
`public final static Color CYAN`	0, 255, 255
`public final static Color MAGENTA`	255, 0, 255
`public final static Color YELLOW`	255, 255, 0
`public final static Color BLACK`	0, 0, 0
`public final static Color WHITE`	255, 255, 255
`public final static Color GRAY`	128, 128, 128
`public final static Color LIGHT_GRAY`	192, 192, 192
`public final static Color DARK_GRAY`	64, 64, 64

Class Color

Method	Description
`Color` *constructors and methods*	
`public Color( int r, int g, int b )`	Creates a color based on red, green and blue components expressed as integers from 0 to 255.
`public Color( float r, float g, float b )`	Creates a color based on red, green and blue components expressed as floating-point values from 0.0 to 1.0.
`public int getRed()`	Returns a value between 0 and 255 representing the red content.
`public int getGreen()`	Returns a value between 0 and 255 representing the green content.
`public int getBlue()`	Returns a value between 0 and 255 representing the blue content.

Class Color

Method	Description
`Graphics` *methods for manipulating* `Colors`	
`public Color getColor()`	Returns `Color` object representing current color for the graphics context.
`public void setColor( Color c )`	Sets the current color for drawing with the graphics context.

Color Control

- Every color is created from a red, a green and a blue component.
  - RGB values: Integers in the range from 0 to 255, or floating-point values in the range 0.0 to 1.0.
  - Specifies the amount of red, the second the amount of green and the third the amount of blue.
  - Larger values == more of that particular color.
  - Approximately 16.7 million colors.
- Graphics method getColor returns a Color object representing the current drawing color.
- Graphics method setColor sets the current drawing color.

Color Control (cont)

- Graphics method fillRect draws a filled rectangle in the current color.
- Four arguments:
  - The first two integer values represent the upper-left x-coordinate and upper-left y-coordinate, where the Graphics object begins drawing the rectangle.
  - The third and fourth arguments are nonnegative integers that represent the width and the height of the rectangle in pixels, respectively.
- A rectangle drawn using method fillRect is filled by the current color of the Graphics object.
- Graphics method drawString draws a String in the current color.

Color Example

```
1  // Fig. 15.5: ColorJPanel.java
2  // Demonstrating Colors.
3  import java.awt.Graphics;
4  import java.awt.Color;
5  import javax.swing.JPanel;
6
7  public class ColorJPanel extends JPanel
8  {
9     // draw rectangles and Strings in different colors
10    public void paintComponent( Graphics g )
11    {
12       super.paintComponent( g ); // call superclass's paintComponent
13
14       this.setBackground( Color.WHITE );
15
16       // set new drawing color using integers
17       g.setColor( new Color( 255, 0, 0 ) );
18       g.fillRect( 15, 25, 100, 20 );
19       g.drawString( "Current RGB: " + g.getColor(), 130, 40 );
20
```

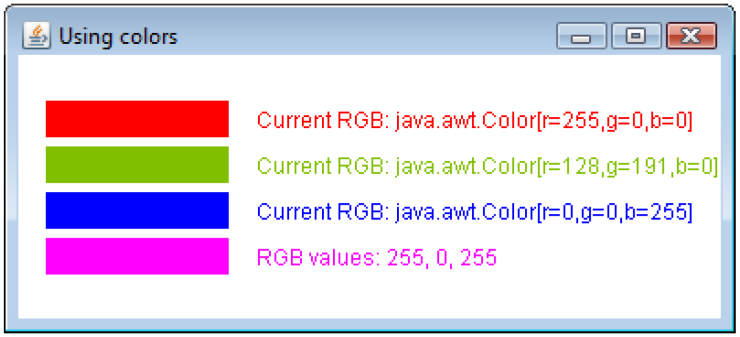
Color Example

```
21          // set new drawing color using floats
22          g.setColor( new Color( 0.50f, 0.75f, 0.0f ) );
23          g.fillRect( 15, 50, 100, 20 );
24          g.drawString( "Current RGB: " + g.getColor(), 130, 65 );
25
26          // set new drawing color using static Color objects
27          g.setColor( Color.BLUE );
28          g.fillRect( 15, 75, 100, 20 );
29          g.drawString( "Current RGB: " + g.getColor(), 130, 90 );
30
31          // display individual RGB values
32          Color color = Color.MAGENTA;
33          g.setColor( color );
34          g.fillRect( 15, 100, 100, 20 );
35          g.drawString( "RGB values: " + color.getRed() + ", " +
36             color.getGreen() + ", " + color.getBlue(), 130, 115 );
37       } // end method paintComponent
38    } // end class ColorJPanel
```

Color Example

```
1  // Fig. 15.6: ShowColors.java
2  // Demonstrating Colors.
3  import javax.swing.JFrame;
4
5  public class ShowColors
6  {
7     // execute application
8     public static void main( String[] args )
9     {
10       // create frame for ColorJPanel
11       JFrame frame = new JFrame( "Using colors" );
12       frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13
14       ColorJPanel colorJPanel = new ColorJPanel(); // create ColorJPanel
15       frame.add( colorJPanel ); // add colorJPanel to frame
16       frame.setSize( 400, 180 ); // set frame size
17       frame.setVisible( true ); // display frame
18    } // end main
19 } // end class ShowColors
```

Color Example



Color Control (cont)

- Package javax.swing provides the JColorChooser GUI component that enables application users to select colors.
- JColorChooser static method showDialog creates a JColorChooser object, attaches it to a dialog box and displays the dialog.
  - Returns the selected Color object, or null if the user presses Cancel or closes the dialog without pressing OK.
  - Three arguments—a reference to its parent Component, a String to display in the title bar of the dialog and the initial selected Color for the dialog.
- Method setBackground changes the background color of a Component.

Color Chooser Example

```
1  // Fig. 15.7: ShowColors2JFrame.java
2  // Choosing colors with JColorChooser.
3  import java.awt.BorderLayout;
4  import java.awt.Color;
5  import java.awt.event.ActionEvent;
6  import java.awt.event.ActionListener;
7  import javax.swing.JButton;
8  import javax.swing.JFrame;
9  import javax.swing.JColorChooser;
10 import javax.swing.JPanel;
11
12 public class ShowColors2JFrame extends JFrame
13 {
14    private JButton changeColorJButton;
15    private Color color = Color.LIGHT_GRAY;
16    private JPanel colorJPanel;
17
```

Color Chooser Example

```
18      // set up GUI
19      public ShowColors2JFrame()
20      {
21         super( "Using JColorChooser" );
22
23         // create JPanel for display color
24         colorJPanel = new JPanel();
25         colorJPanel.setBackground( color );
26
27         // set up changeColorJButton and register its event handler
28         changeColorJButton = new JButton( "Change Color" );
29         changeColorJButton.addActionListener(
30
31            new ActionListener() // anonymous inner class
32            {
33               // display JColorChooser when user clicks button
34               public void actionPerformed( ActionEvent event )
35               {
36                  color = JColorChooser.showDialog(
37                     ShowColors2JFrame.this, "Choose a color", color );
38
```

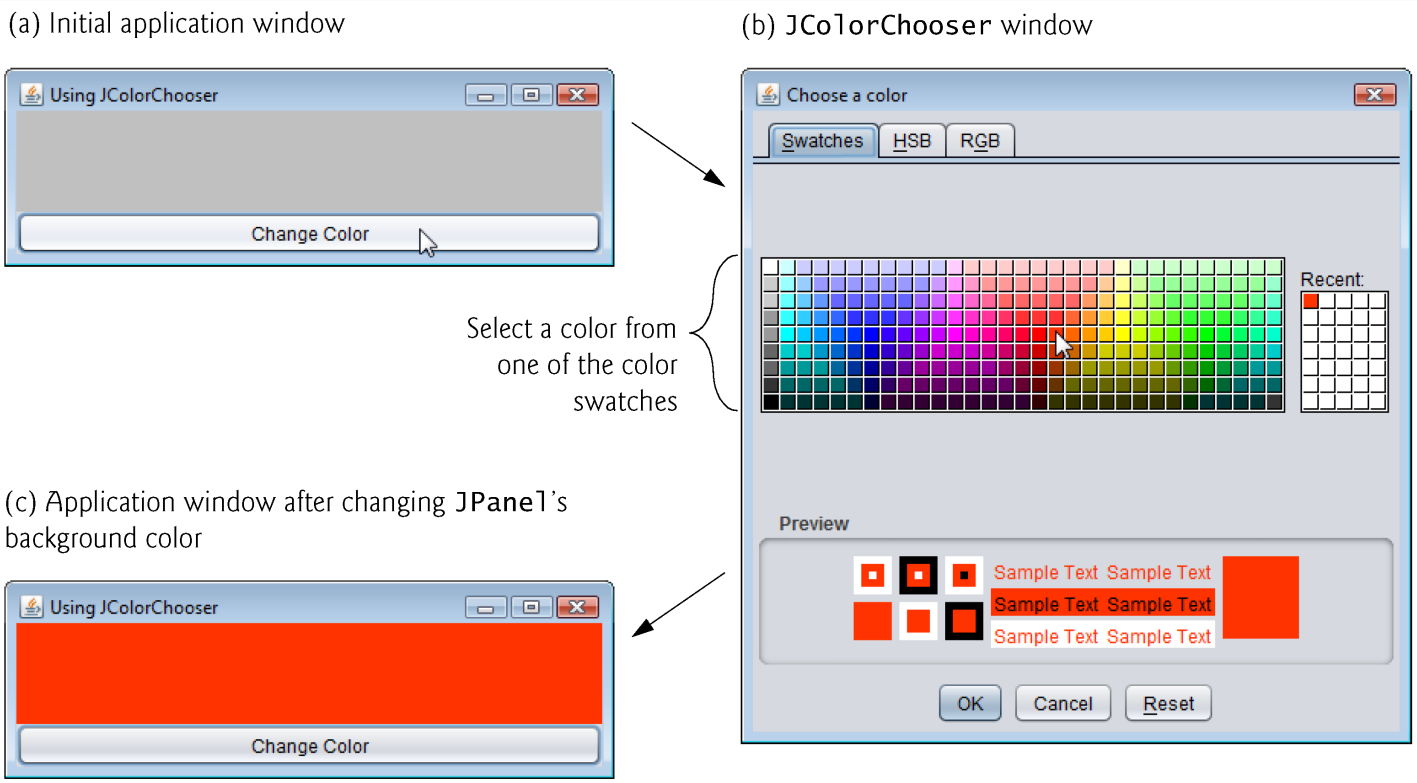
Color Chooser Example

```
39                   // set default color, if no color is returned
40                   if ( color == null )
41                      color = Color.LIGHT_GRAY;
42
43                   // change content pane's background color
44                   colorJPanel.setBackground( color );
45                } // end method actionPerformed
46             } // end anonymous inner class
47          ); // end call to addActionListener
48
49          add( colorJPanel, BorderLayout.CENTER ); // add colorJPanel
50          add( changeColorJButton, BorderLayout.SOUTH ); // add button
51
52          setSize( 400, 130 ); // set frame size
53          setVisible( true ); // display frame
54       } // end ShowColor2JFrame constructor
55    } // end class ShowColors2JFrame
```

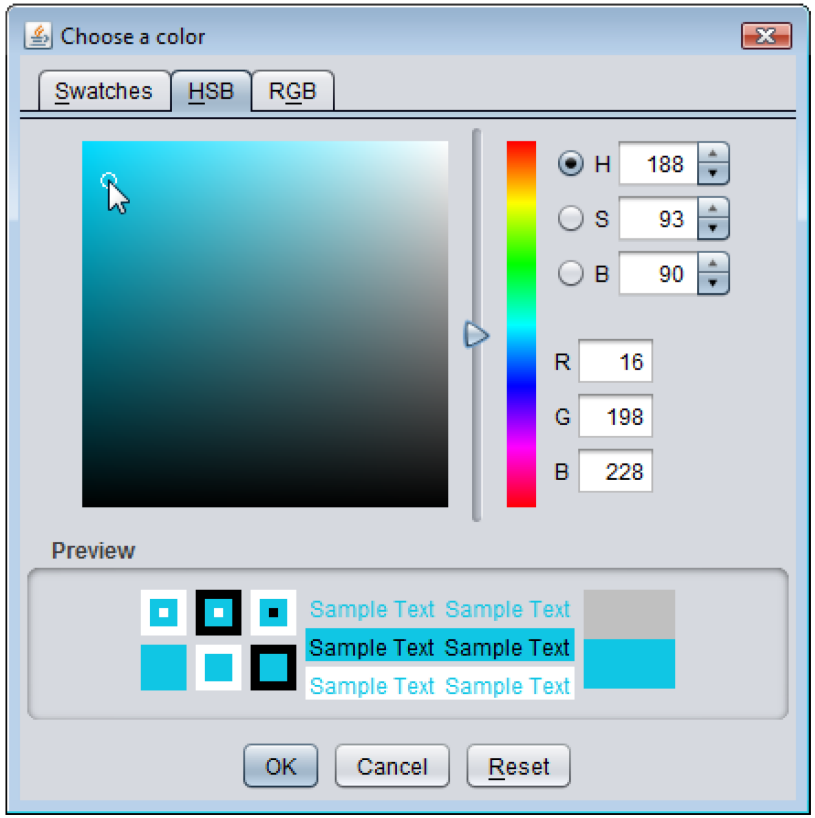
Color Chooser Example

```
1  // Fig. 15.8: ShowColors2.java
2  // Choosing colors with JColorChooser.
3  import javax.swing.JFrame;
4
5  public class ShowColors2
6  {
7     // execute application
8     public static void main( String[] args )
9     {
10       ShowColors2JFrame application = new ShowColors2JFrame();
11       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
12    } // end main
13 } // end class ShowColors2
```

Color Chooser Example

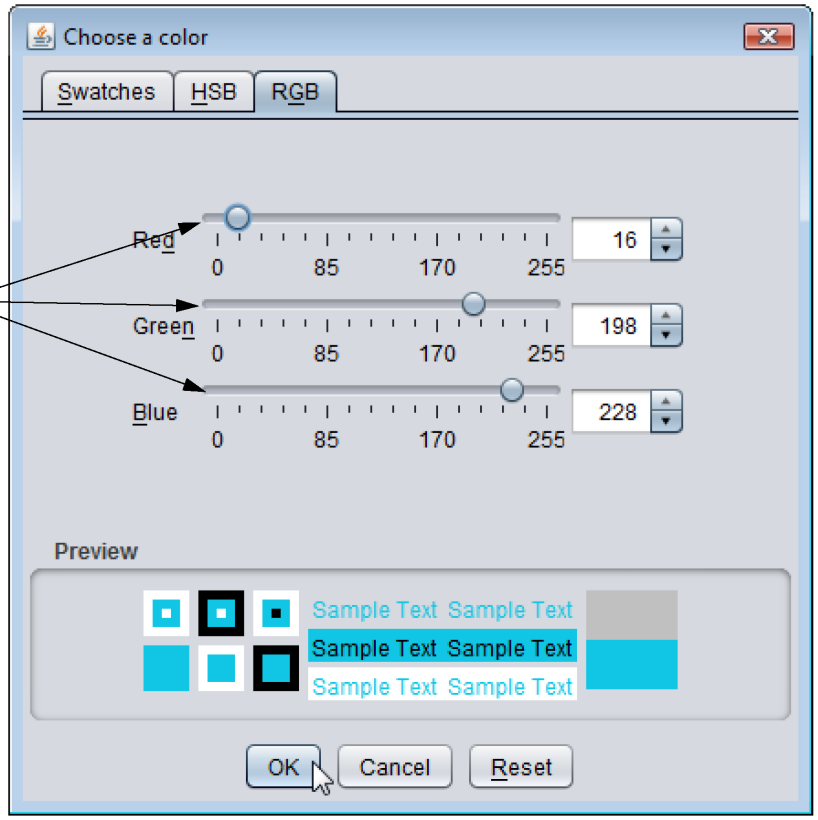


Color Chooser Example



Color Chooser Example

Sliders to select the red, green and blue color components



Manipulating Fonts

- Most font methods and font constants are part of class Font.
- Some methods of class Font and class Graphics are summarized in Fig. 15.10.
- Class Font's constructor takes three arguments—the font name, font style and font size.
  - Any font currently supported by the system on which the program is running, such as standard Java fonts Monospaced, SansSerif and Serif.
  - The font style is Font.PLAIN, Font.ITALIC or Font.BOLD.
  - Font styles can be used in combination.
- The font size is measured in points.
  - A point is 1/72 of an inch.
- Graphics method setFont sets the current drawing font—the font in which text will be displayed—to its Font argument.

Manipulating Fonts

Method or constant	Description
`Font` *constants, constructors and methods*	
`public final static int PLAIN`	A constant representing a plain font style.
`public final static int BOLD`	A constant representing a bold font style.
`public final static int ITALIC`	A constant representing an italic font style.
`public Font( String name, int style, int size )`	Creates a `Font` object with the specified font name, style and size.
`public int getStyle()`	Returns an `int` indicating the current font style.
`public int getSize()`	Returns an `int` indicating the current font size.
`public String getName()`	Returns the current font name as a string.
`public String getFamily()`	Returns the font's family name as a string.
`public boolean isPlain()`	Returns `true` if the font is plain, else `false`.
`public boolean isBold()`	Returns `true` if the font is bold, else `false`.
`public boolean isItalic()`	Returns `true` if the font is italic, else `false`.

Manipulating Fonts

Method or constant	Description
`Graphics` *methods for manipulating* `Font`s	
`public Font getFont()`	Returns a `Font` object reference representing the current font.
`public void setFont( Font f )`	Sets the current font to the font, style and size specified by the `Font` object reference `f`.

Font Example

```
1  // Fig. 15.11: FontJPanel.java
2  // Display strings in different fonts and colors.
3  import java.awt.Font;
4  import java.awt.Color;
5  import java.awt.Graphics;
6  import javax.swing.JPanel;
7
8  public class FontJPanel extends JPanel
9  {
10    // display Strings in different fonts and colors
11    public void paintComponent( Graphics g )
12    {
13       super.paintComponent( g ); // call superclass's paintComponent
14
```

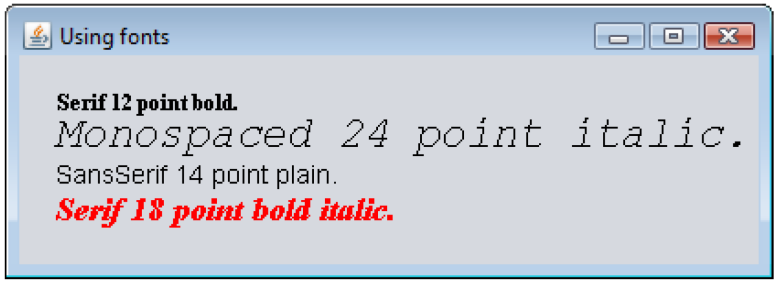
Font Example

```
15          // set font to Serif (Times), bold, 12pt and draw a string
16          g.setFont( new Font( "Serif", Font.BOLD, 12 ) );
17          g.drawString( "Serif 12 point bold.", 20, 30 );
18
19          // set font to Monospaced (Courier), italic, 24pt and draw a string
20          g.setFont( new Font( "Monospaced", Font.ITALIC, 24 ) );
21          g.drawString( "Monospaced 24 point italic.", 20, 50 );
22
23          // set font to SansSerif (Helvetica), plain, 14pt and draw a string
24          g.setFont( new Font( "SansSerif", Font.PLAIN, 14 ) );
25          g.drawString( "SansSerif 14 point plain.", 20, 70 );
26
27          // set font to Serif (Times), bold/italic, 18pt and draw a string
28          g.setColor( Color.RED );
29          g.setFont( new Font( "Serif", Font.BOLD + Font.ITALIC, 18 ) );
30          g.drawString( g.getFont().getName() + " " + g.getFont().getSize() +
31             " point bold italic.", 20, 90 );
32       } // end method paintComponent
33    } // end class FontJPanel
```

Font Example

```
1  // Fig. 15.12: Fonts.java
2  // Using fonts.
3  import javax.swing.JFrame;
4
5  public class Fonts
6  {
7     // execute application
8     public static void main( String[] args )
9     {
10       // create frame for FontJPanel
11       JFrame frame = new JFrame( "Using fonts" );
12       frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13
14       FontJPanel fontJPanel = new FontJPanel(); // create FontJPanel
15       frame.add( fontJPanel ); // add fontJPanel to frame
16       frame.setSize( 420, 150 ); // set frame size
17       frame.setVisible( true ); // display frame
18    } // end main
19 } // end class Fonts
```

Font Example



Drawing Lines, Rectangles, Ovals

- This section presents Graphics methods for drawing lines, rectangles and ovals.
- The methods and their parameters are summarized in Fig. 15.17.

Drawing Lines, Rectangles, Ovals

Method	Description
`public void drawLine( int x1, int y1, int x2, int y2 )`	Draws a line between the point (`x1`, `y1`) and the point (`x2`, `y2`).
`public void drawRect( int x, int y, int width, int height )`	Draws a rectangle of the specified `width` and `height`. The rectangle's top-left corner is located at (`x`, `y`). Only the outline of the rectangle is drawn using the `Graphics` object's color—the body of the rectangle is not filled with this color.
`public void fillRect( int x, int y, int width, int height )`	Draws a filled rectangle in the current color with the specified `width` and `height`. The rectangle's top-left corner is located at (`x`, `y`).
`public void clearRect( int x, int y, int width, int height )`	Draws a filled rectangle with the specified `width` and `height` in the current background color. The rectangle's top-left corner is located at (`x`, `y`). This method is useful if you want to remove a portion of an image.

Drawing Lines, Rectangles, Ovals

Method	Description
`public void drawRoundRect( int x, int y, int width, int height, int arcWidth, int arcHeight )`	Draws a rectangle with rounded corners in the current color with the specified `width` and `height`. The `arcWidth` and `arcHeight` determine the rounding of the corners (see Fig. 15.20). Only the outline of the shape is drawn.
`public void fillRoundRect( int x, int y, int width, int height, int arcWidth, int arcHeight )`	Draws a filled rectangle in the current color with rounded corners with the specified `width` and `height`. The `arcWidth` and `arcHeight` determine the rounding of the corners (see Fig. 15.20).
`public void draw3DRect( int x, int y, int width, int height, boolean b )`	Draws a three-dimensional rectangle in the current color with the specified `width` and `height`. The rectangle's top-left corner is located at `(x, y)`. The rectangle appears raised when `b` is true and lowered when `b` is false. Only the outline of the shape is drawn.

Drawing Lines, Rectangles, Ovals

Method	Description
`public void fill3DRect( int x, int y, int width, int height, boolean b )`	Draws a filled three-dimensional rectangle in the current color with the specified `width` and `height`. The rectangle's top-left corner is located at (`x`, `y`). The rectangle appears raised when `b` is true and lowered when `b` is false.
`public void drawOval( int x, int y, int width, int height )`	Draws an oval in the current color with the specified `width` and `height`. The bounding rectangle's top-left corner is located at (`x`, `y`). The oval touches all four sides of the bounding rectangle at the center of each side (see Fig. 15.21). Only the outline of the shape is drawn.
`public void fillOval( int x, int y, int width, int height )`	Draws a filled oval in the current color with the specified `width` and `height`. The bounding rectangle's top-left corner is located at (`x`, `y`). The oval touches the center of all four sides of the bounding rectangle (see Fig. 15.21).

Drawing Lines, Rectangles, Ovals Example

```
1  // Fig. 15.18: LinesRectsOvalsJPanel.java
2  // Drawing lines, rectangles and ovals.
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import javax.swing.JPanel;
6
7  public class LinesRectsOvalsJPanel extends JPanel
8  {
9     // display various lines, rectangles and ovals
10    public void paintComponent( Graphics g )
11    {
12       super.paintComponent( g ); // call superclass's paint method
13
14       this.setBackground( Color.WHITE );
15
```

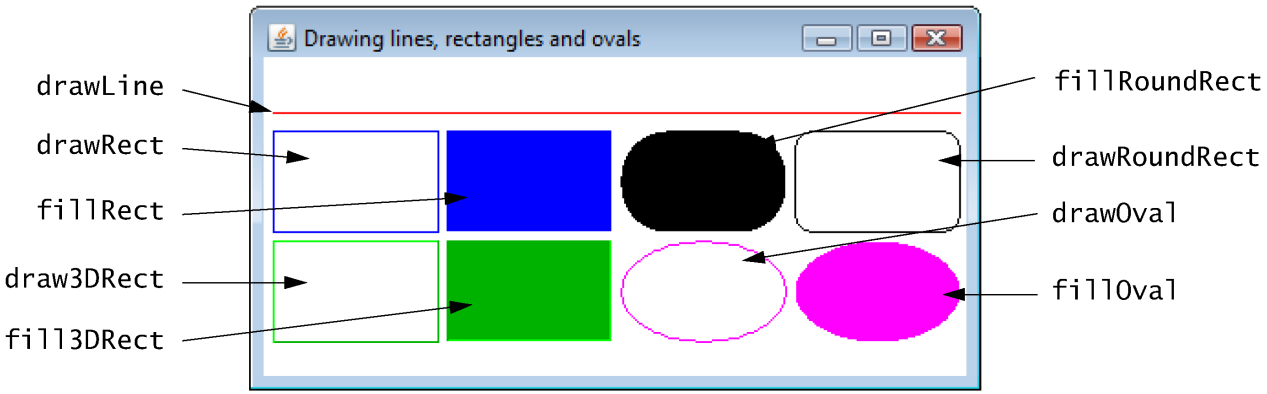
Drawing Lines, Rectangles, Ovals Example

```
16          g.setColor( Color.RED );
17          g.drawLine( 5, 30, 380, 30 );
18
19          g.setColor( Color.BLUE );
20          g.drawRect( 5, 40, 90, 55 );
21          g.fillRect( 100, 40, 90, 55 );
22
23          g.setColor( Color.CYAN );
24          g.fillRoundRect( 195, 40, 90, 55, 50, 50 );
25          g.drawRoundRect( 290, 40, 90, 55, 20, 20 );
26
27          g.setColor( Color.GREEN );
28          g.draw3DRect( 5, 100, 90, 55, true );
29          g.fill3DRect( 100, 100, 90, 55, false );
30
31          g.setColor( Color.MAGENTA );
32          g.drawOval( 195, 100, 90, 55 );
33          g.fillOval( 290, 100, 90, 55 );
34       } // end method paintComponent
35    } // end class LinesRectsOvalsJPanel
```

Drawing Lines, Rectangles, Ovals Example

```
1  // Fig. 15.19: LinesRectsOvals.java
2  // Drawing lines, rectangles and ovals.
3  import java.awt.Color;
4  import javax.swing.JFrame;
5
6  public class LinesRectsOvals
7  {
8     // execute application
9     public static void main( String[] args )
10    {
11       // create frame for LinesRectsOvalsJPanel
12       JFrame frame =
13          new JFrame( "Drawing lines, rectangles and ovals" );
14       frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
15
16       LinesRectsOvalsJPanel linesRectsOvalsJPanel =
17          new LinesRectsOvalsJPanel();
18       linesRectsOvalsJPanel.setBackground( Color.WHITE );
19       frame.add( linesRectsOvalsJPanel ); // add panel to frame
20       frame.setSize( 400, 210 ); // set frame size
21       frame.setVisible( true ); // display frame
22    } // end main
23 } // end class LinesRectsOvals
```

Drawing Lines, Rectangles, Ovals Example



Drawing Polygons

- Polygons are closed multisided shapes composed of straight-line segments.
- Polylines are sequences of connected points.
- Some methods require a Polygon object (package java.awt).

Drawing Polygons

Method	Description
`Graphics` methods for drawing polygons	
`public void drawPolygon( int[] xPoints, int[] yPoints, int points )`	Draws a polygon. The *x*-coordinate of each point is specified in the `xPoints` array and the *y*-coordinate of each point in the `yPoints` array. The last argument specifies the number of `points`. This method draws a closed polygon. If the last point is different from the first, the polygon is closed by a line that connects the last point to the first.
`public void drawPolyline( int[] xPoints, int[] yPoints, int points )`	Draws a sequence of connected lines. The *x*-coordinate of each point is specified in the `xPoints` array and the *y*-coordinate of each point in the `yPoints` array. The last argument specifies the number of `points`. If the last point is different from the first, the polyline is not closed.
`public void drawPolygon( Polygon p )`	Draws the specified polygon.

Drawing Polygons

Method	Description
`public void fillPolygon( int[] xPoints, int[] yPoints, int points )`	Draws a filled polygon. The *x*-coordinate of each point is specified in the `xPoints` array and the *y*-coordinate of each point in the `yPoints` array. The last argument specifies the number of `points`. This method draws a closed polygon. If the last point is different from the first, the polygon is closed by a line that connects the last point to the first.
`public void fillPolygon( Polygon p )`	Draws the specified filled polygon. The polygon is closed.

Drawing Polygons

Method	Description
`Polygon` constructors and methods	
`public Polygon()`	Constructs a new polygon object. The polygon does not contain any points.
`public Polygon( int[] xValues, int[] yValues, int numberOfPoints )`	Constructs a new polygon object. The polygon has `numberOfPoints` sides, with each point consisting of an *x*-coordinate from `xValues` and a *y*-coordinate from `yValues`.
`public void addPoint( int x, int y )`	Adds pairs of *x*- and *y*-coordinates to the `Polygon`.

Drawing Polygons Example

```
1  // Fig. 15.27: PolygonsJPanel.java
2  // Drawing polygons.
3  import java.awt.Graphics;
4  import java.awt.Polygon;
5  import javax.swing.JPanel;
6
7  public class PolygonsJPanel extends JPanel
8  {
9     // draw polygons and polylines
10    public void paintComponent( Graphics g )
11    {
12       super.paintComponent( g ); // call superclass's paintComponent
13
14       // draw polygon with Polygon object
15       int[] xValues = { 20, 40, 50, 30, 20, 15 };
16       int[] yValues = { 50, 50, 60, 80, 80, 60 };
17       Polygon polygon1 = new Polygon( xValues, yValues, 6 );
18       g.drawPolygon( polygon1 );
19
```

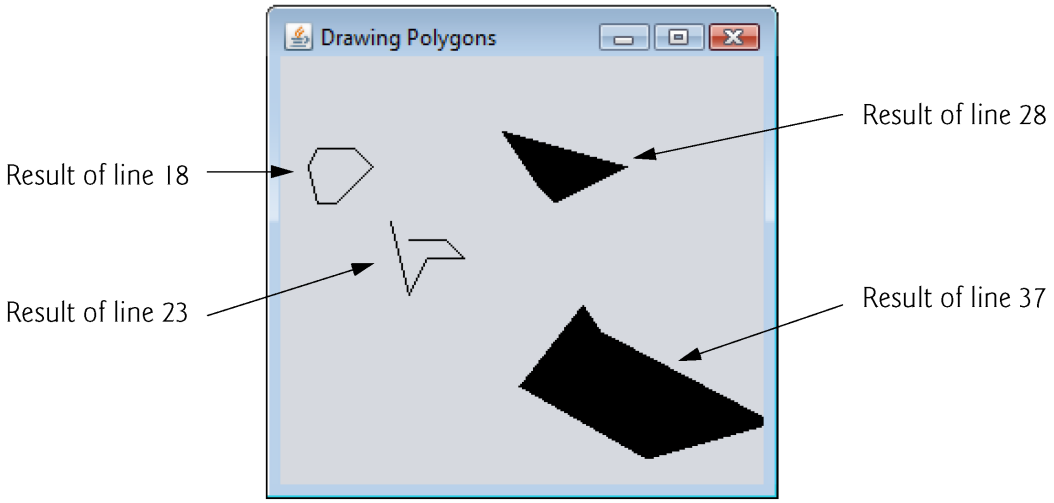
Drawing Polygons Example

```
20          // draw polylines with two arrays
21          int[] xValues2 = { 70, 90, 100, 80, 70, 65, 60 };
22          int[] yValues2 = { 100, 100, 110, 110, 130, 110, 90 };
23          g.drawPolyline( xValues2, yValues2, 7 );
24
25          // fill polygon with two arrays
26          int[] xValues3 = { 120, 140, 150, 190 };
27          int[] yValues3 = { 40, 70, 80, 60 };
28          g.fillPolygon( xValues3, yValues3, 4 );
29
30          // draw filled polygon with Polygon object
31          Polygon polygon2 = new Polygon();
32          polygon2.addPoint( 165, 135 );
33          polygon2.addPoint( 175, 150 );
34          polygon2.addPoint( 270, 200 );

35          polygon2.addPoint( 200, 220 );
36          polygon2.addPoint( 130, 180 );
37          g.fillPolygon( polygon2 );
38       } // end method paintComponent
39    } // end class PolygonsJPanel
```

Drawing Polygons Example

```
1  // Fig. 15.28: DrawPolygons.java
2  // Drawing polygons.
3  import javax.swing.JFrame;
4
5  public class DrawPolygons
6  {
7     // execute application
8     public static void main( String[] args )
9     {
10       // create frame for PolygonsJPanel
11       JFrame frame = new JFrame( "Drawing Polygons" );
12       frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13
14       PolygonsJPanel polygonsJPanel = new PolygonsJPanel();
15       frame.add( polygonsJPanel ); // add polygonsJPanel to frame
16       frame.setSize( 280, 270 ); // set frame size
17       frame.setVisible( true ); // display frame
18    } // end main
19 } // end class DrawPolygons
```

Drawing Polygons Example



Animating Drawing Objects

- In the Java GUI application, we can create a timer and make it generate events at certain interval
- In the following, the timer calls the actionPerformed function of the current class after every 100ms

```
Timer time = new Timer(100, this);
time.start();
```

Animating Drawing Objects

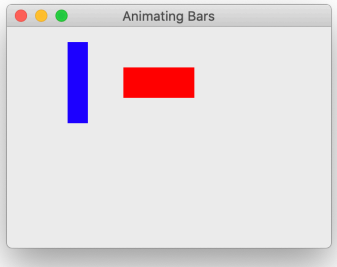
- We can create a JPanel and include a timer in it, that will call the actionPerformed periodically.
  - in the actionPeformed function, we would update the location of the drawing objects

```
public BarAnimation(){
      p1 = new Point(20, 40);
      p2 = new Point(60, 10);

      dx = dy = 5;
      // set up timer to animate rectangles every 100 ms
      Timer time = new Timer(100, this);
      time.start();
}
```

Animating Drawing Objects

- The paintComponent function draws two filled rectangle at p1 and p2 positions respectively as the following:

```
// draws two rectangles on this panel on the screen
    public void paintComponent(Graphics g) {

    super.paintComponent(g); // call JPanel's version
    g.setColor(Color.RED);
    g.fillRect(p1.x, p1.y, 70, 30);
    g.setColor(Color.BLUE);
    g.fillRect(p2.x, p2.y, 20, 80);
    }
```



Animating Drawing Objects

- In the actionPerfomed function, we update the location of the filledRects

```
// Handles clicks on Compute button by computing the BMI.
public void actionPerformed(ActionEvent event) {
      p1.x += dx;
      p2.y += dy;
      if (p1.x <= 0 || p1.x + 70 >= getWidth()) {
            dx = - dx; // rectangle 1 has hit left/right edge
      }
      if (p2.y <= 0 || p2.y + 80 >= getHeight()) {
            dy = - dy; // rectangle 2 has hit top/bottom edge
      }
      repaint(); // updates the drawing
}
```

Thank you

Please let me know if you have any further questions!