JOptionPane

Java JOptionPane

- The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box.

- These dialog boxes are used to display information or get input from the user. The JOptionPane class inherits JComponent class.

- A dialog is normally used as a temporary window to receive additional information from the user, or to provide notification that some event has occurred.

- Java provides the JOptionPane class, which can be used to create standard dialogs. You can also build custom dialogs by extending the JDialog class.

Types of JOptionPanes

- public static void showMessageDialog(Component parent, Object message) Displays a message on a dialog with an OK button.

- public static int showConfirmDialog(Component parent, Object message) Displays a message and list of choices Yes, No, Cancel.

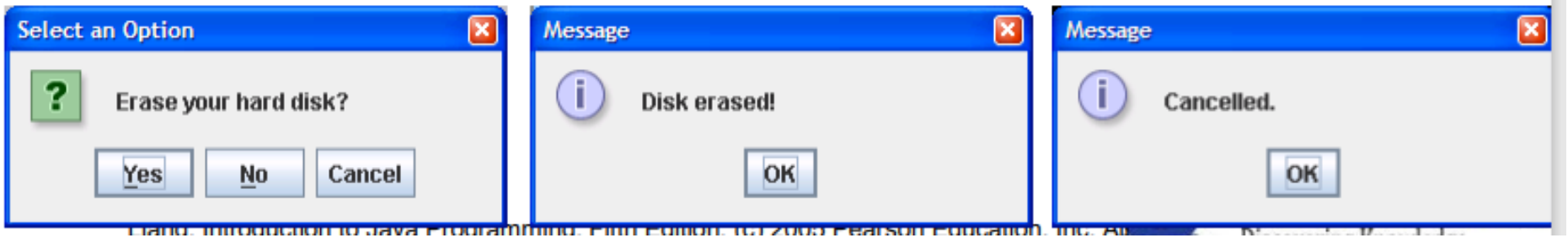
- public static String showInputDialog(Component parent, Object message) Displays a message and text field for input, and returns the value entered as a String.

JOptionPane examples 1

```
import javax.swing.*;
class MessageDialogExample {
 public static void main(String[] args) {
 JOptionPane.showMessageDialog(null,
 "How's the weather?");
 JOptionPane.showMessageDialog(null,
 "Second message");
 }
}
```

JOptionPane examples 2

```
import javax.swing.*;
class ConfirmDialogExample {
 public static void main(String[] args) {
 int choice = JOptionPane.showConfirmDialog(null,
 "Erase your hard disk?");
 if (choice == JOptionPane.YES_OPTION) {
 JOptionPane.showMessageDialog(null, "Disk erased!");
 } else {
 JOptionPane.showMessageDialog(null, "Cancelled.");
 }
 }
}
```



JOptionPane examples 3

```
import javax.swing.*;
class InputDialogExample {
 public static void main(String[] args) {
 String name = JOptionPane.showInputDialog(null,
 "What’s your name?");
 JOptionPane.showMessageDialog(null, “Hello " + name);
 }
}
```

Message Dialogs

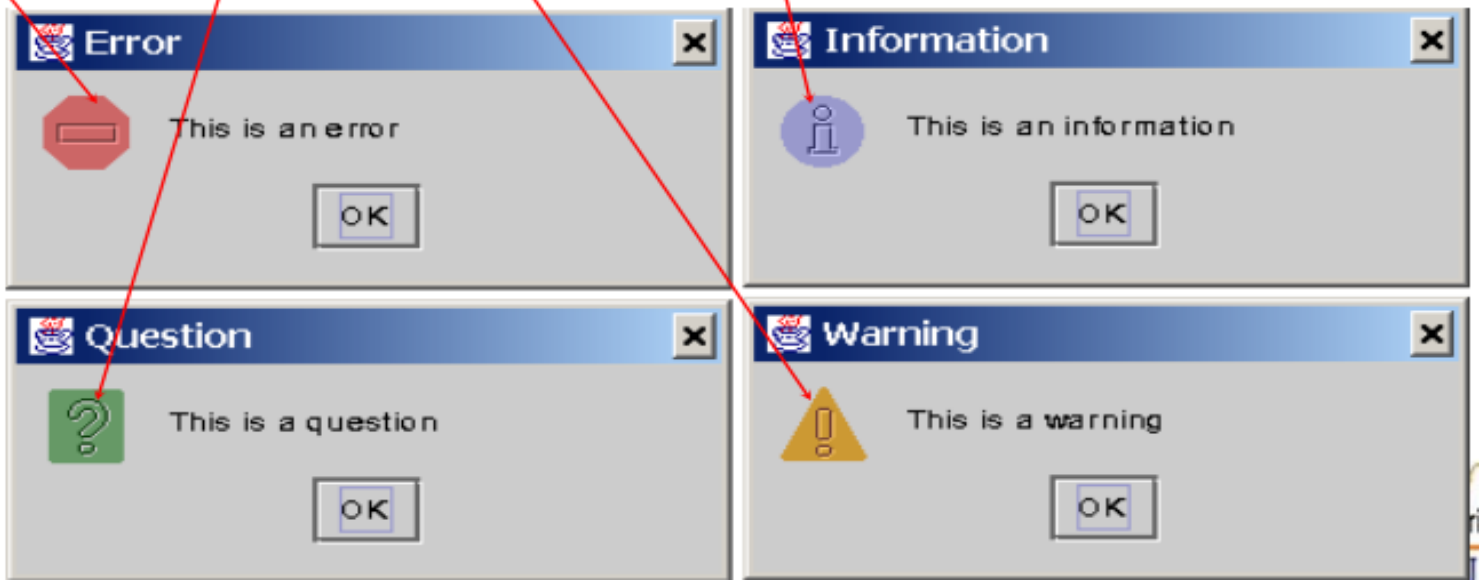
- A message dialog box simply displays a message to alert the user and waits for the user to click the OK button to close the dialog.

```
JOptionPane.showMessageDialog(null, “This is an error",
  “Error", JOptionPane.INFORMATION_MESSAGE);
```

Message Types

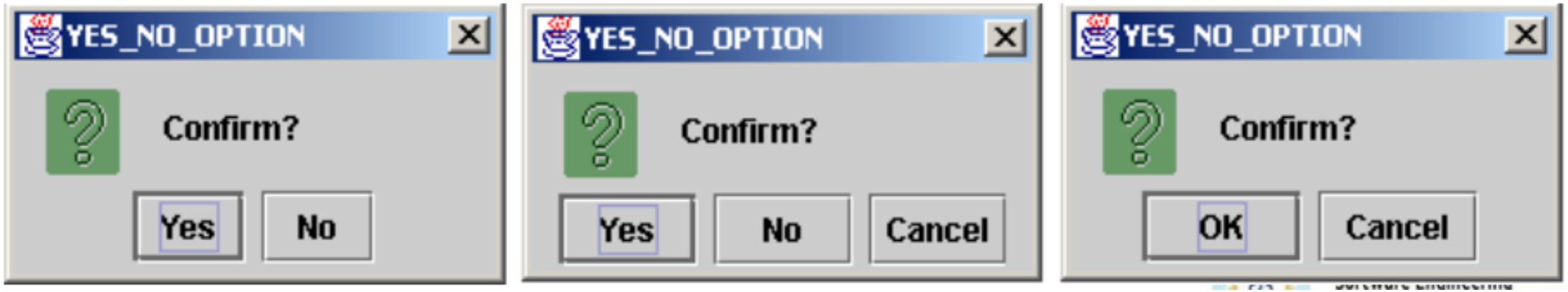
The messageType is one of the following constants:

- JOptionPane.ERROR_MESSAGE
- JOptionPane.INFORMATION_MESSAGE
- JOptionPane.PLAIN_MESSAGE
- JOptionPane.WARNING_MESSAGE
- JOptionPane.QUESTION_MESSAGE



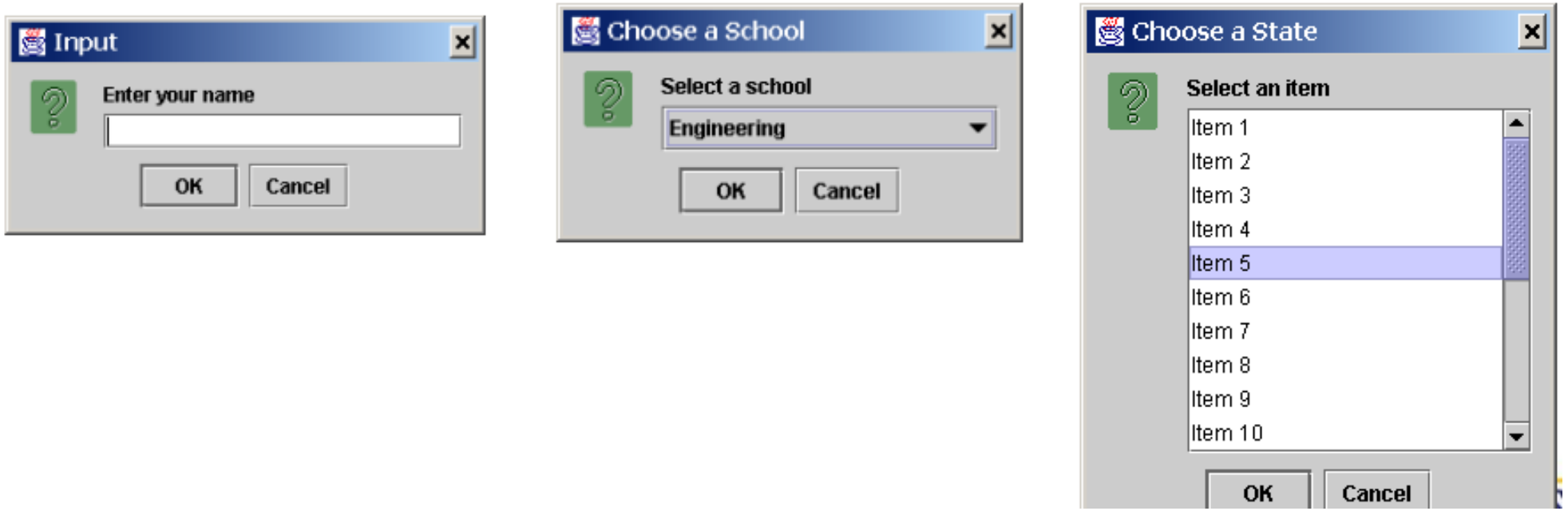
Confirmation Dialogs

- A message dialog box displays a message and waits for the user to click the OK button to dismiss the dialog. The message dialog does not return any value. A confirmation dialog asks a question and requires the user to respond with an appropriate button. The confirmation dialog returns a value that corresponds to a selected button.



Input Dialogs

- An input dialog box is used to receive input from the user. The input can be entered from a text field or selected from a combo box or a list. Selectable values can be specified in an array, and a particular value can be designated as the initial selected value.



Option Dialogs

An *option dialog* allows you to create custom buttons.

example

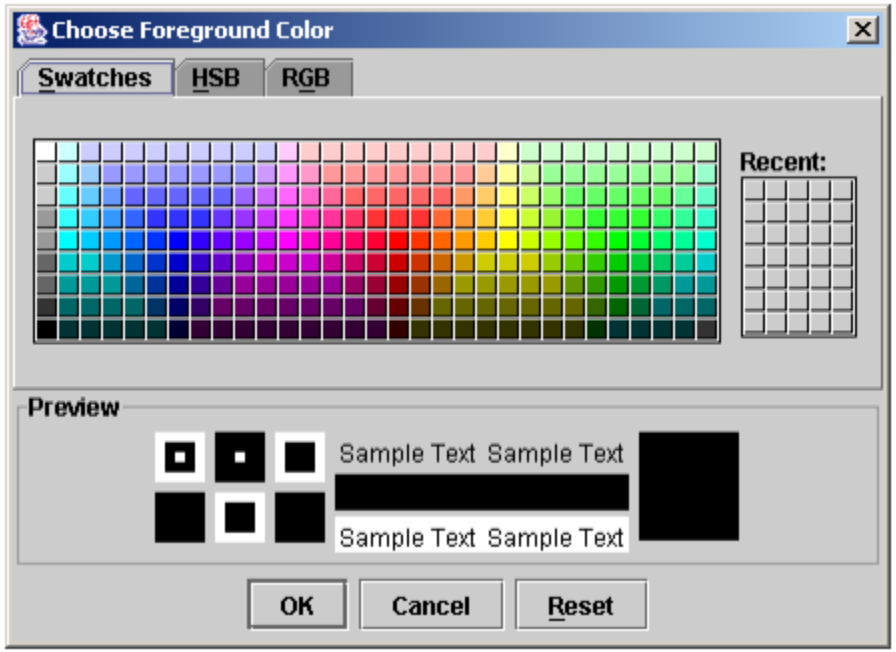
```
import javax.swing.JFrame;
import javax.swing.JOptionPane;


public class MessageDialogExample {
    public static void main(String[] args) {
        // Create a JFrame
        JFrame frame = new JFrame();
        frame.setTitle("My Frame Title");


        // Get the frame title
        String title = frame.getTitle();


        // Display the message dialog box
        JOptionPane.showMessageDialog(frame, title, "Message", JOptionPane.INFORMATION_MESSAGE);
    }
}
```

JColorChooser

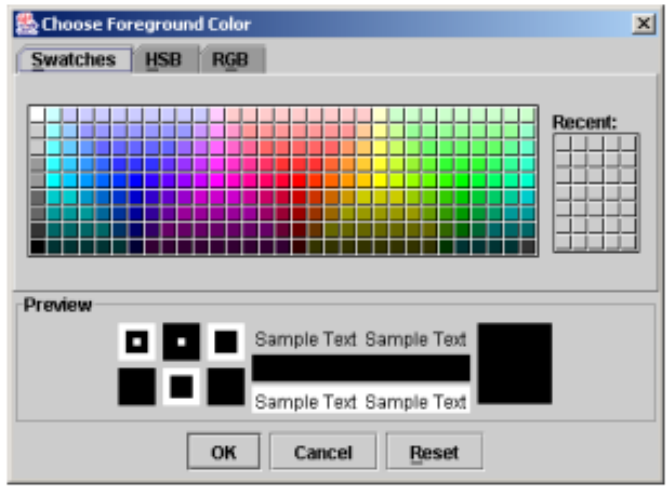
```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
public class ColorChooserExample extends JFrame implements ActionListener {
JButton b;
Container c;
ColorChooserExample(){
    c=getContentPane();
    c.setLayout(new FlowLayout());
    b=new JButton("color");
    b.addActionListener(this);
    c.add(b);
}
public void actionPerformed(ActionEvent e) {
Color initialcolor=Color.RED;
Color color=JColorChooser.showDialog(this,"Select a color",initialcolor);
c.setBackground(color);
}

public static void main(String[] args) {
    ColorChooserExample ch=new ColorChooserExample();
    ch.setSize(400,400);
    ch.setVisible(true);
    ch.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
}
```

JColorChooser

- Color dialogs are commonly used in GUI programming. Swing provides a convenient and versatile color dialog named javax.swing.JColorChooser. Like JOptionPane, JColorChooser is a lightweight component inherited from JComponent. It can be added to any container



Using JColorChooser



To create a JColorChooser, use

```
new JColorChooser();
```

To display a JColorChooser dialog box, use

```
public static Color showDialog(Component parentComponent,
   String title,  Color initialColor)
```

This method creates an instance of JDialog with three buttons, OK, Cancel, and Reset, to hold a JColorChooser object, as shown in Figure 29.27. The method displays a modal dialog. If the user clicks the *OK* button, the method dismisses the dialog and returns the selected color. If the user clicks the *Cancel* button or closes the dialog, the method dismisses the dialog and returns null.

Java JColorChooser Example

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
public class ColorChooserExample extends JFrame implements ActionListener {
JButton b;
Container c;
ColorChooserExample(){
    c=getContentPane();
    c.setLayout(new FlowLayout());
    b=new JButton("color");
    b.addActionListener(this);
    c.add(b);
}
public void actionPerformed(ActionEvent e) {
Color initialcolor=Color.RED;
Color color=JColorChooser.showDialog(this,"Select a color",initialcolor);
c.setBackground(color);
}

public static void main(String[] args) {
    ColorChooserExample ch=new ColorChooserExample();
    ch.setSize(400,400);
    ch.setVisible(true);
    ch.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
}
```

Font Control

- Class `Font`
  - Constructor takes three arguments—the font name, font style and font size
    - Font name – any font currently supported by the system on which the program is running
    - Font style –`Font.PLAIN`, `Font.ITALIC` or `Font.BOLD`. Font styles can be used in combination
    - Font sizes – measured in points. A point is 1/72 of an inch.
  - Methods `getName`, `getStyle` and `getSize` retrieve information about `Font` object
  - `Graphics` methods `getFont` and `setFont` retrieve and set the current font, respectively

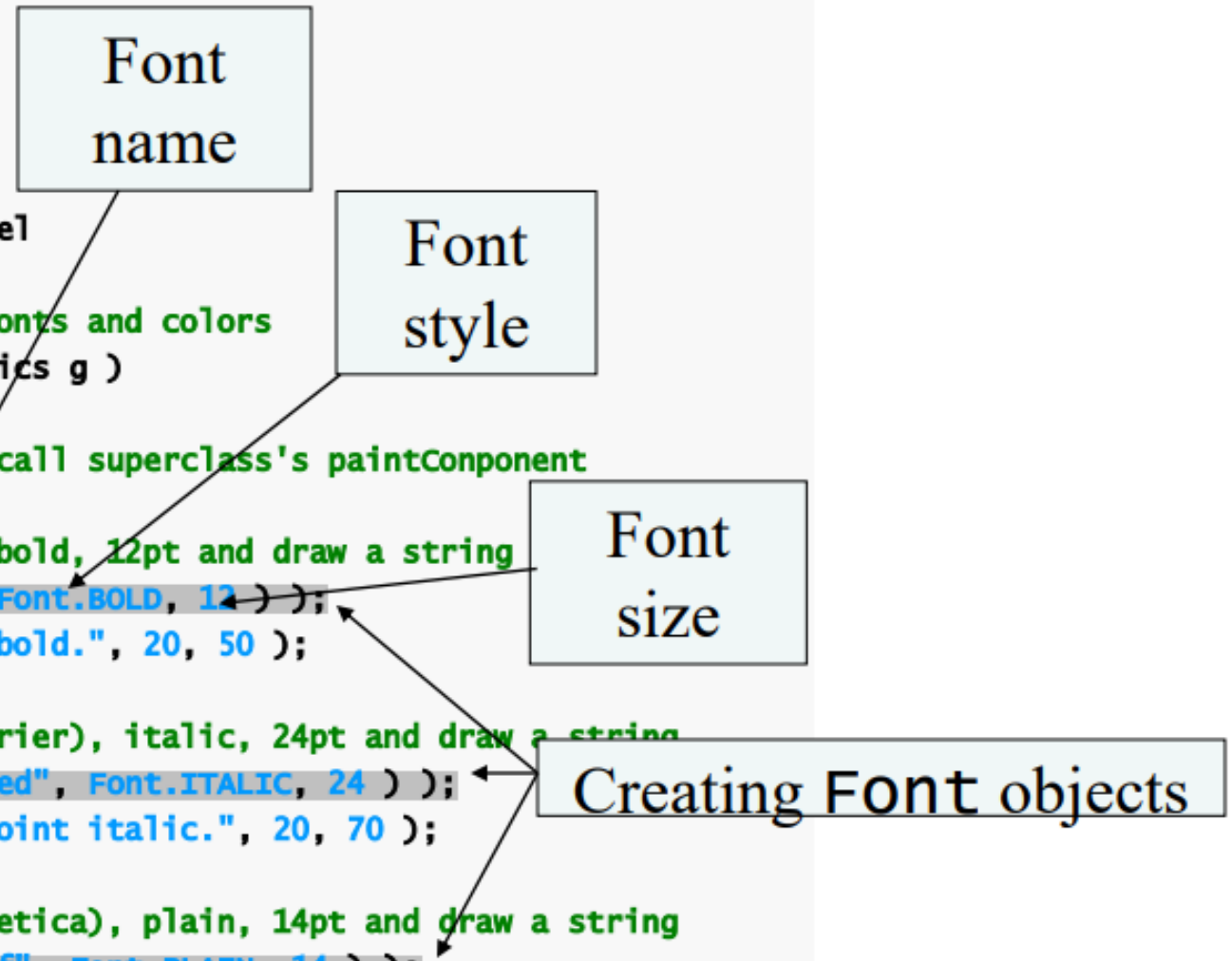
Font-related methods and constants. (Part 1 of 2)

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 integer value indicating the current font style.
`public int getSize()`	Returns an integer value indicating the current font size.

Font-related methods and constants. (Part 2 of 2)

Method or constant	Description
`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`.
`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`.

```java
// Fig. 12.11: FontJPanel.java
// Display strings in different fonts and colors.
import java.awt.Font;
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

public class FontJPanel extends JPanel
{
   // display Strings in different fonts and colors
   public void paintComponent( Graphics g )
   {
      super.paintComponent( g ); // call superclass's paintConponent

      // set font to Serif (Times), bold, 12pt and draw a string
      g.setFont( new Font( "Serif", Font.BOLD, 12 ) );
      g.drawString( "Serif 12 point bold.", 20, 50 );

      // set font to Monospaced (Courier), italic, 24pt and draw a string
      g.setFont( new Font( "Monospaced", Font.ITALIC, 24 ) );
      g.drawString( "Monospaced 24 point italic.", 20, 70 );

      // set font to SansSerif (Helvetica), plain, 14pt and draw a string
      g.setFont( new Font( "SansSerif", Font.PLAIN, 14 ) );
      g.drawString( "SansSerif 14 point plain.", 20, 90 );
```



Combining styles

```
    // set font to Serif (Times), bold/italic, 18pt and draw a string
    g.setColor( Color.RED );
    g.setFont( new Font( "Serif", Font.BOLD + Font.ITALIC, 18 ) );
    g.drawString( g.getFont().getName() + " " + g.getFont().getSize() +
       " point bold italic.", 20, 110 );
 } // end method paintComponent
// end class FontJPanel
```

Retrieve font name and size of `Graphics` object's current `Font`

```java
// Fig. 12.12: Fonts.java
// Using fonts.
import javax.swing.JFrame;

public class Fonts
{
   // execute application
   public static void main( String args[] )
   {
      // create frame for FontJPanel
      JFrame frame = new JFrame( "Using fonts" );
      frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

      FontJPanel fontJPanel = new FontJPanel(); // create FontJPanel
      frame.add( fontJPanel ); // add fontJPanel to frame
      frame.setSize( 420, 170 ); // set frame size
      frame.setVisible( true ); // display frame
   } // end main
} // end class Fonts
```