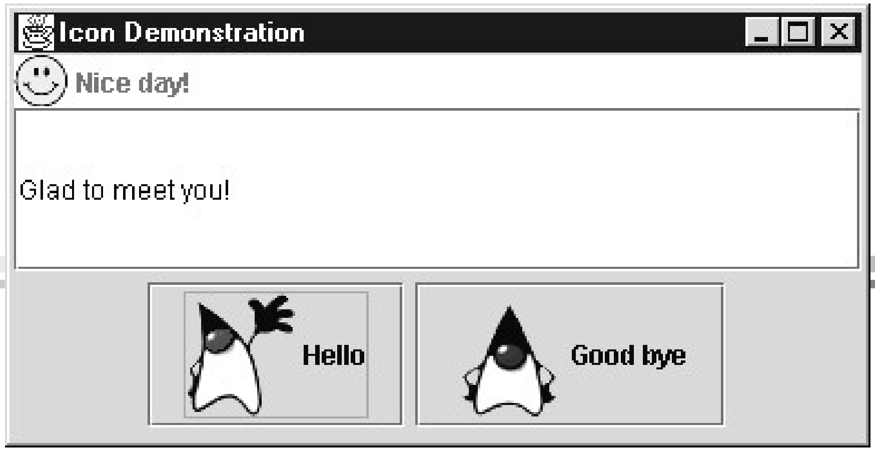
15.2 Making GUIs Pretty (and More Functional)

- Adding Icons
- The `JScrollPane` Class for Scroll Bars
- Adding Borders
- Changing the Look and Feel

Using Icons



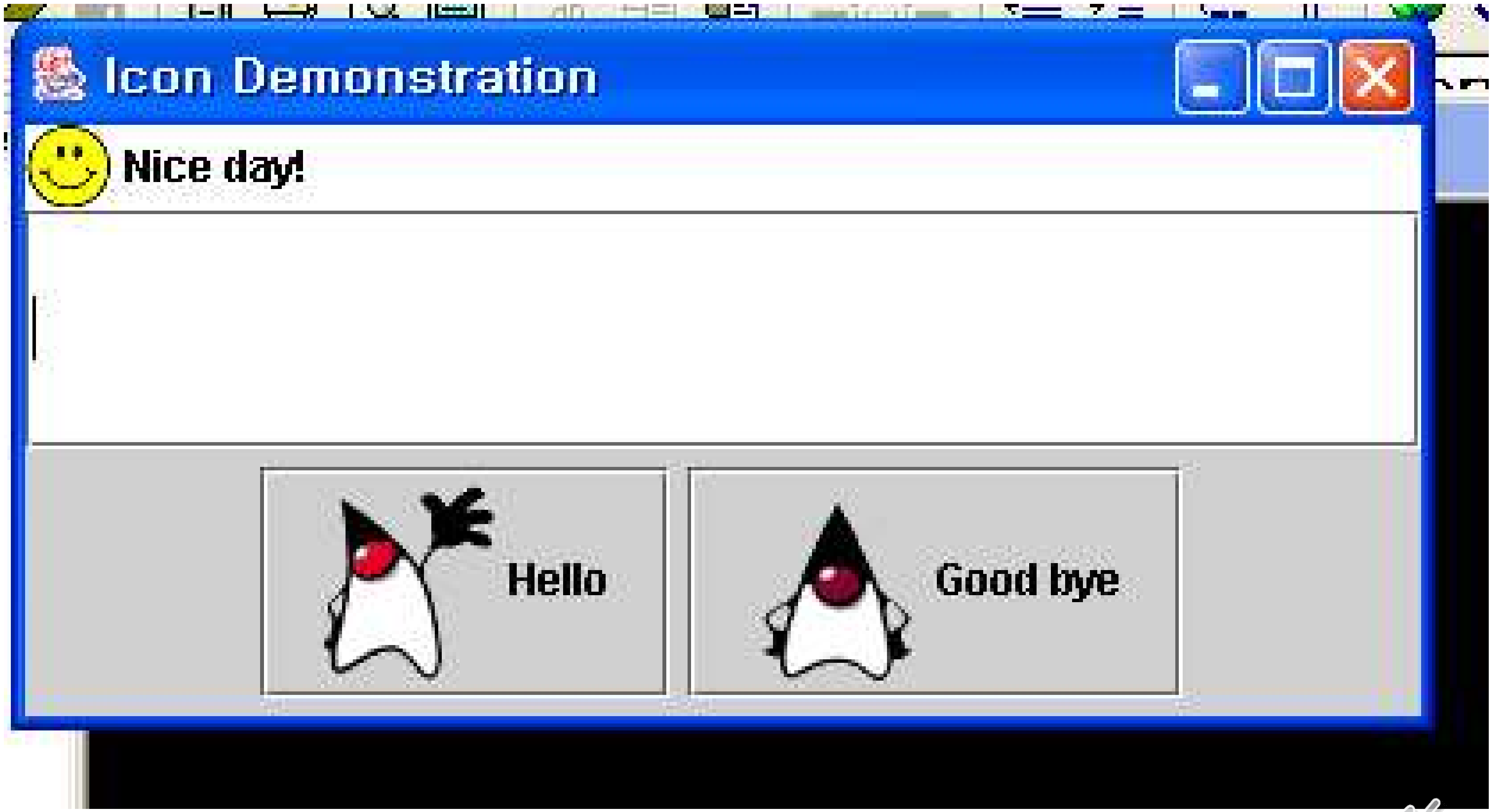
Glad to meet you!

- Icons are (small) pictures
- Icons may be added to labels, buttons, and menu items.
- The `ImageIcon` class can be used to convert a picture to an icon:

```
ImageIcon SmileyFaceIcon = new ImageIcon(“smiley.gif”);
```

- The `setIcon` method can be used to add an icon to a component:

```
JLabel helloLabel = new JLabel(“Hello”);
ImageIcon dukeWavingIcon =
        new ImageIcon(“duke_waving.gif”);
helloLabel.setIcon(dukeWavingIcon);
```


Icon Demonstration
Nice day!
Hello
Good bye

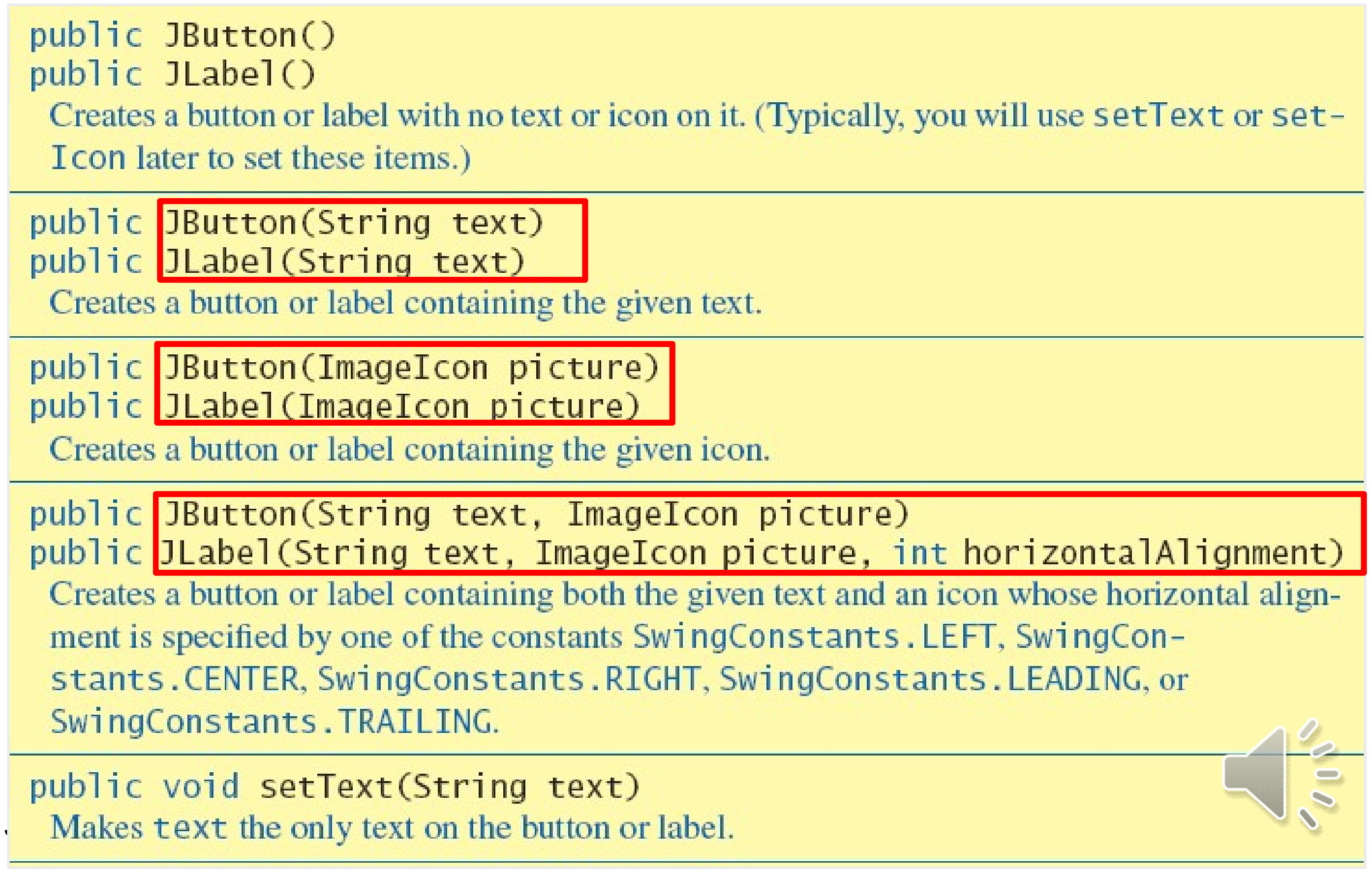
Listing 15.2 Using Icons IconDemo.java

```
// Listing.2 Using Icons
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/**
 Simple demonstration of putting icons in buttons and labels.
*/
public class IconDemo extends JFrame implements ActionListener
{
    public static final int WIDTH = 400;
    public static final int HEIGHT = 200;

    private JTextField message;

    public IconDemo( )
    {
        setSize(WIDTH, HEIGHT);
        addWindowListener(new WindowDestroyer( ));
        setTitle("Icon Demonstration");
        Container content = getContentPane( );
        content.setBackground(Color.WHITE);
        content.setLayout(new BorderLayout( ));
                    //
    JLabel niceLabel = new JLabel("Nice day!");
    ImageIcon smileyIcon = new ImageIcon("smiley.gif");
    niceLabel.setIcon(smileyIcon);
    content.add(niceLabel, BorderLayout.NORTH);

    JPanel buttonPanel = new JPanel( );
    buttonPanel.setLayout(new FlowLayout( ));
    //
    JButton helloButton = new JButton("Hello");
    ImageIcon dukeWavingIcon = new ImageIcon("duke_waving.gif");
    helloButton.setIcon(dukeWavingIcon);
    helloButton.addActionListener(this);
    buttonPanel.add(helloButton);
    //
    JButton byeButton = new JButton("Good bye");
    ImageIcon dukeStandingIcon =
            new ImageIcon("duke_standing.gif");
    byeButton.setIcon(dukeStandingIcon);
    byeButton.addActionListener(this);
    buttonPanel.add(byeButton);
    content.add(buttonPanel, BorderLayout.SOUTH);

    message = new JTextField(30);
    content.add(message, BorderLayout.CENTER);
}
```

```
 public void actionPerformed(ActionEvent e)
   {
      if (e.getActionCommand( ).equals("Hello"))
          message.setText("Glad to meet you!");
      else if (e.getActionCommand( ).equals("Good bye"))
          message.setText(
                 "OK, click the upper right button. I'll miss you.");
      else
          System.out.println("Error in button interface.");
   }

   /**
    Creates and displays a window of the class IconDemo.
   */
   public static void main(String[] args)
   {
       IconDemo iconGui = new IconDemo( );
       iconGui.setVisible(true);
   }
}
```

Swing Methods

- Figure 15.1a some methods in the classes **`JButton`** and **`JLabel`**



`public JButton()`
`public JLabel()`
Creates a button or label with no text or icon on it. (Typically, you will use `setText` or `setIcon` later to set these items.)

`public JButton(String text)`
`public JLabel(String text)`
Creates a button or label containing the given text.

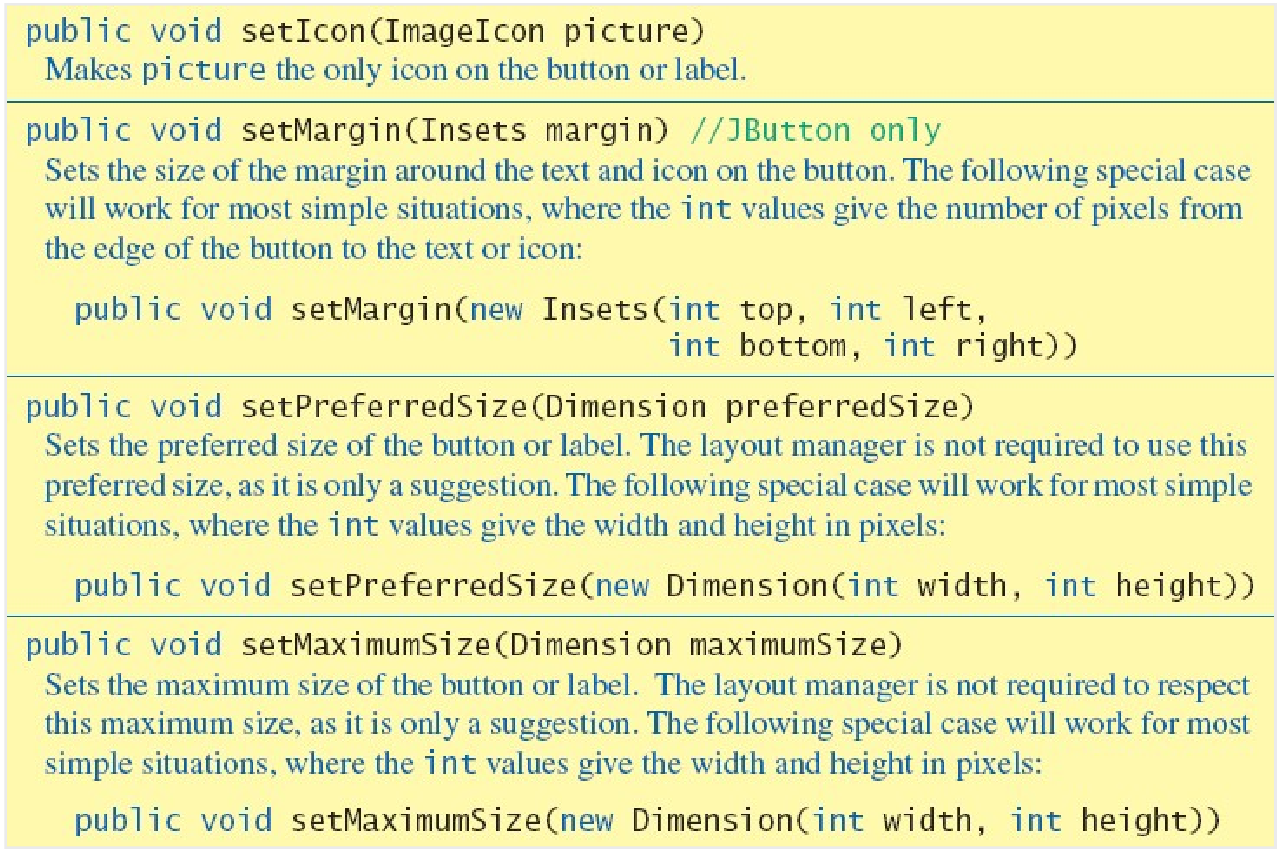
`public JButton(ImageIcon picture)`
`public JLabel(ImageIcon picture)`
Creates a button or label containing the given icon.

`public JButton(String text, ImageIcon picture)`
`public JLabel(String text, ImageIcon picture, int horizontalAlignment)`
Creates a button or label containing both the given text and an icon whose horizontal alignment is specified by one of the constants `SwingConstants.LEFT`, `SwingConstants.CENTER`, `SwingConstants.RIGHT`, `SwingConstants.LEADING`, or `SwingConstants.TRAILING`.

`public void setText(String text)`
Makes `text` the only text on the button or label.

Swing Methods

- Figure 15.1b some methods in the classes `JButton` and `JLabel`



```
public void setIcon(ImageIcon picture)
```
Makes `picture` the only icon on the button or label.

```
public void setMargin(Insets margin) //JButton only
```
Sets the size of the margin around the text and icon on the button. The following special case will work for most simple situations, where the `int` values give the number of pixels from the edge of the button to the text or icon:

```
public void setMargin(new Insets(int top, int left,
                                 int bottom, int right))
```

```
public void setPreferredSize(Dimension preferredSize)
```
Sets the preferred size of the button or label. The layout manager is not required to use this preferred size, as it is only a suggestion. The following special case will work for most simple situations, where the `int` values give the width and height in pixels:

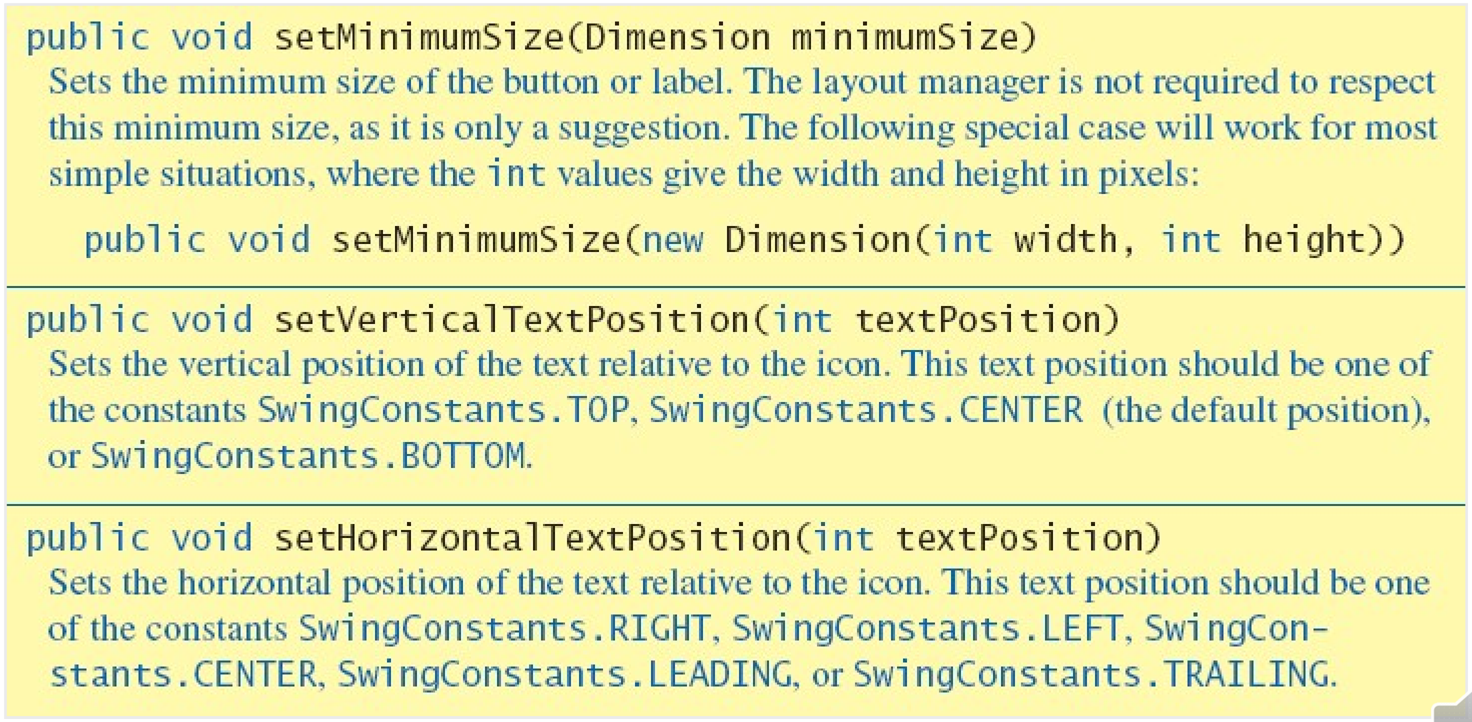
```
public void setPreferredSize(new Dimension(int width, int height))
```

```
public void setMaximumSize(Dimension maximumSize)
```
Sets the maximum size of the button or label. The layout manager is not required to respect this maximum size, as it is only a suggestion. The following special case will work for most simple situations, where the `int` values give the width and height in pixels:

```
public void setMaximumSize(new Dimension(int width, int height))
```

Swing Methods

- Figure 15.1c some methods in the classes `JButton` and `JLabel`



`public void setMinimumSize(Dimension minimumSize)`
Sets the minimum size of the button or label. The layout manager is not required to respect this minimum size, as it is only a suggestion. The following special case will work for most simple situations, where the `int` values give the width and height in pixels:

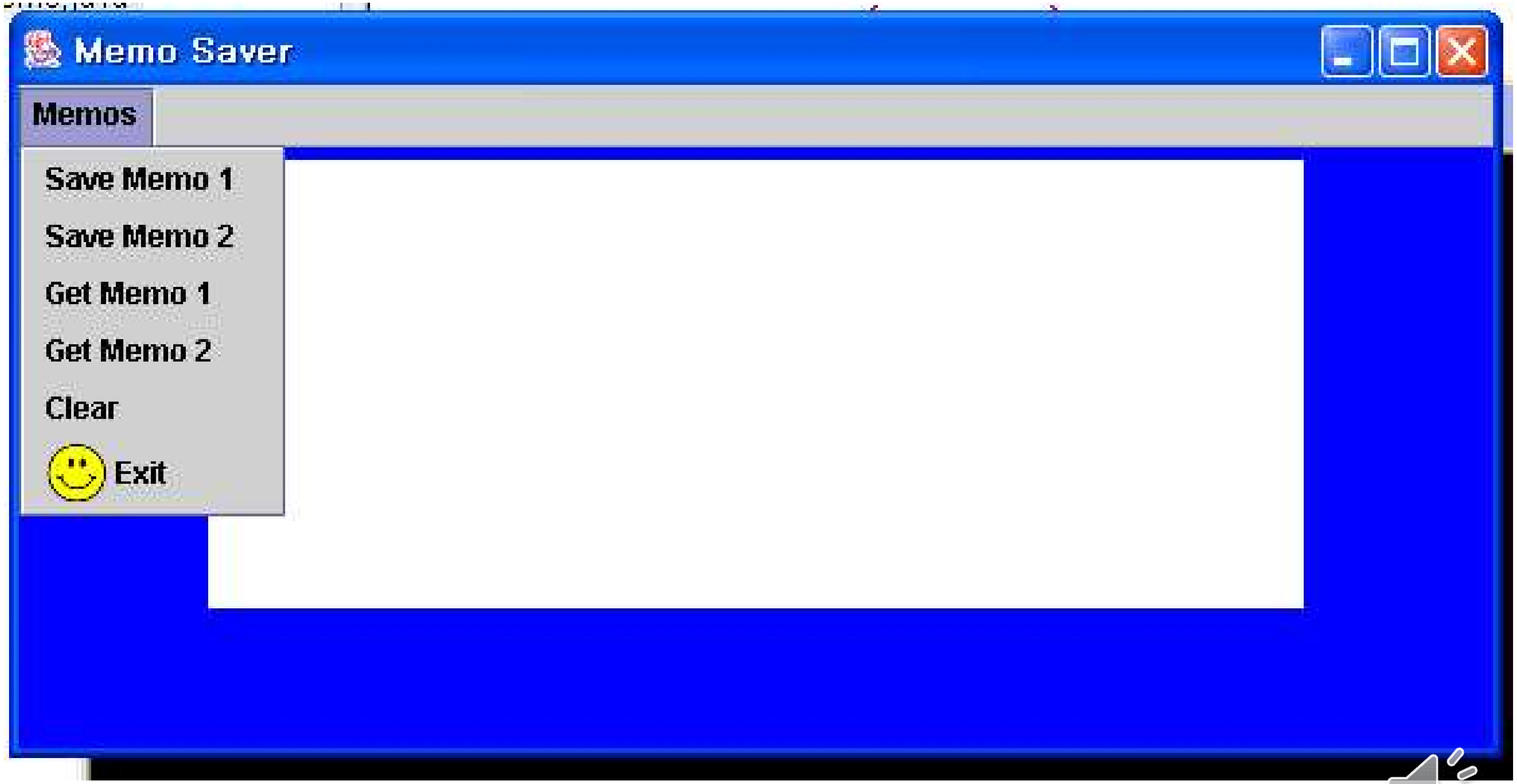
```
public void setMinimumSize(new Dimension(int width, int height))
```

`public void setVerticalTextPosition(int textPosition)`
Sets the vertical position of the text relative to the icon. This text position should be one of the constants `SwingConstants.TOP`, `SwingConstants.CENTER` (the default position), or `SwingConstants.BOTTOM`.

`public void setHorizontalTextPosition(int textPosition)`
Sets the horizontal position of the text relative to the icon. This text position should be one of the constants `SwingConstants.RIGHT`, `SwingConstants.LEFT`, `SwingCon-stants.CENTER`, `SwingConstants.LEADING`, or `SwingConstants.TRAILING`.

Adding Icons to Menu Items

- setIcon()
  - » to add an icon to a JMenuItem
  - » MemoIconDemo.java


Memo Saver
Memos
Save Memo 1
Save Memo 2
Get Memo 1
Get Memo 2
Clear
Exit

MemoIconDemo.java

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

public class MemoIconDemo extends JFrame implements ActionListener
{
    public static final int WIDTH = 600;
    public static final int HEIGHT = 300;
    public static final int LINES = 10;
    public static final int CHAR_PER_LINE = 40;

    private JTextArea theText;
    private String memo1 = "No Memo 1.";
    private String memo2 = "No Memo 2.";
```

```
public MemoIconDemo()
  {
      setSize(WIDTH, HEIGHT);
      addWindowListener(new WindowDestroyer());
      setTitle("Memo Saver");
      Container contentPane = getContentPane();
      contentPane.setLayout(new BorderLayout());

      JMenu memoMenu = new JMenu("Memos");
      JMenuItem m;

      m = new JMenuItem("Save Memo 1");
      m.addActionListener(this);
      memoMenu.add(m);

      m = new JMenuItem("Save Memo 2");
      m.addActionListener(this);
      memoMenu.add(m);

      m = new JMenuItem("Get Memo 1");
      m.addActionListener(this);
      memoMenu.add(m);
m = new JMenuItem("Get Memo 2");
      m.addActionListener(this);
      memoMenu.add(m);

      m = new JMenuItem("Clear");
      m.addActionListener(this);
      memoMenu.add(m);

      m = new JMenuItem("Exit");
      m.addActionListener(this);
      ImageIcon smileyIcon = new ImageIcon("smiley.gif");
      m.setIcon(smileyIcon);      ///
      memoMenu.add(m);

      JMenuBar mBar = new JMenuBar();
      mBar.add(memoMenu);
      setJMenuBar(mBar);

      JPanel textPanel = new JPanel();
      textPanel.setBackground(Color.BLUE);
      theText = new JTextArea(LINES, CHAR_PER_LINE);
      theText.setBackground(Color.WHITE);
      textPanel.add(theText);
      contentPane.add(textPanel, BorderLayout.CENTER);
   }
```

```
public void actionPerformed(ActionEvent e)
   {
      String actionCommand = e.getActionCommand();
      if (actionCommand.equals("Save Memo 1"))
         memo1 = theText.getText();
      else if (actionCommand.equals("Save Memo 2"))
         memo2 = theText.getText();
      else if (actionCommand.equals("Clear"))
         theText.setText("");
      else if (actionCommand.equals("Get Memo 1"))
         theText.setText(memo1);
      else if (actionCommand.equals("Get Memo 2"))
         theText.setText(memo2);
      else if (actionCommand.equals("Exit"))
         System.exit(0);
      else
         theText.setText("Error in memo interface");
   }

   public static void main(String[] args)
   {
      MemoIconDemo gui = new MemoIconDemo();
      gui.setVisible(true);
   }
}
```

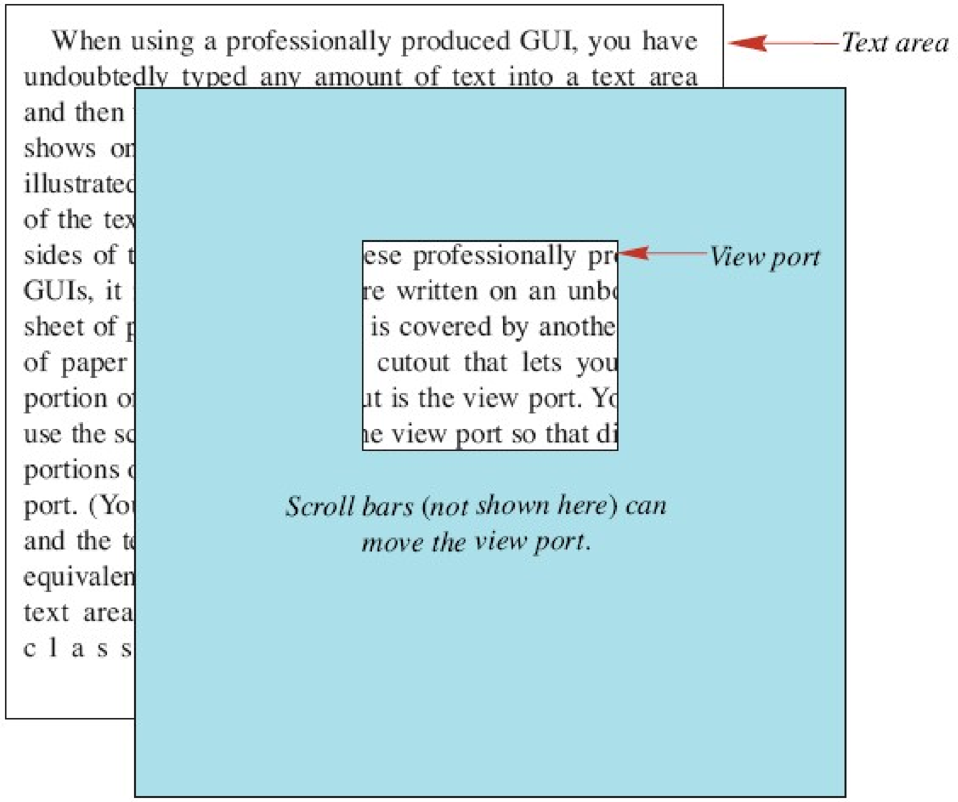
The [redacted] Class for Scroll Bars

- A [redacted]*t* is used when not all information can be displayed on screen at once.
- Scroll bars move a view port around to show different parts of the information.
  - » Fig 15.2 View Port for a Text Area
- `JScrollPane` is a class that can provide a view port with scroll bars.
- An example using `JScrollPane` with a `JTextArea` called `theText` and a `JPanel` called `textPanel`:

```
JScrollPane scrolledText = new JScrollPane(theText);
textPanel.add(scrolledText);
```

Text Area Features

- Figure 15.2 View port for a text area



Listing 15.3 A Text Area with Scroll Bars - ScrollBarDemo.java

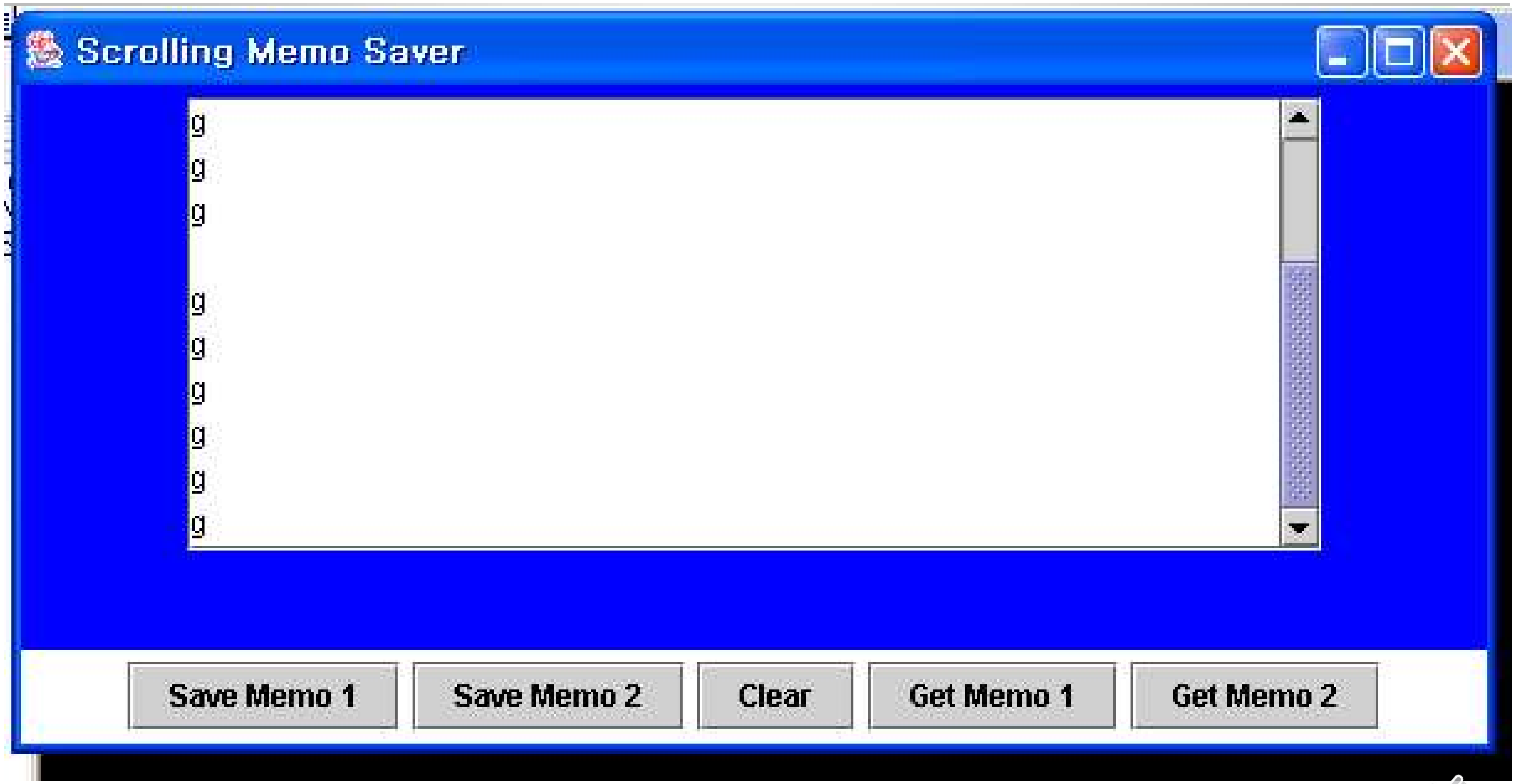
» scrolledText.setHorizontalScrollBarPolicy(
  JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
» scrolledText.setVerticalScrollBarPolicy(
  JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

```
// Listing 15.3  A Text Area with Scroll Bars

import javax.swing.*;

import java.awt.*;
import java.awt.event.*;

public class ScrollBarDemo extends JFrame implements ActionListener
{
    public static final int WIDTH = 600;
    public static final int HEIGHT = 300;
    public static final int LINES = 10;
    public static final int CHAR_PER_LINE = 40;

    private JTextArea theText;
    private String memo1 = "No Memo 1.";
    private String memo2 = "No Memo 2.";
```

```
public ScrollBarDemo( )
  {
     setSize(WIDTH, HEIGHT);
     addWindowListener(new WindowDestroyer( ));
     setTitle("Scrolling Memo Saver");
     Container contentPane = getContentPane( );
     contentPane.setLayout(new BorderLayout( ));

     JPanel buttonPanel = new JPanel( );
     buttonPanel.setBackground(Color.WHITE);
     buttonPanel.setLayout(new FlowLayout( ));
     JButton memo1Button = new JButton("Save Memo 1");
     memo1Button.addActionListener(this);
     buttonPanel.add(memo1Button);
     JButton memo2Button = new JButton("Save Memo 2");
     memo2Button.addActionListener(this);
     buttonPanel.add(memo2Button);
     JButton clearButton = new JButton("Clear");
     clearButton.addActionListener(this);
     buttonPanel.add(clearButton);
     JButton get1Button = new JButton("Get Memo 1");
     get1Button.addActionListener(this);
     buttonPanel.add(get1Button);
```

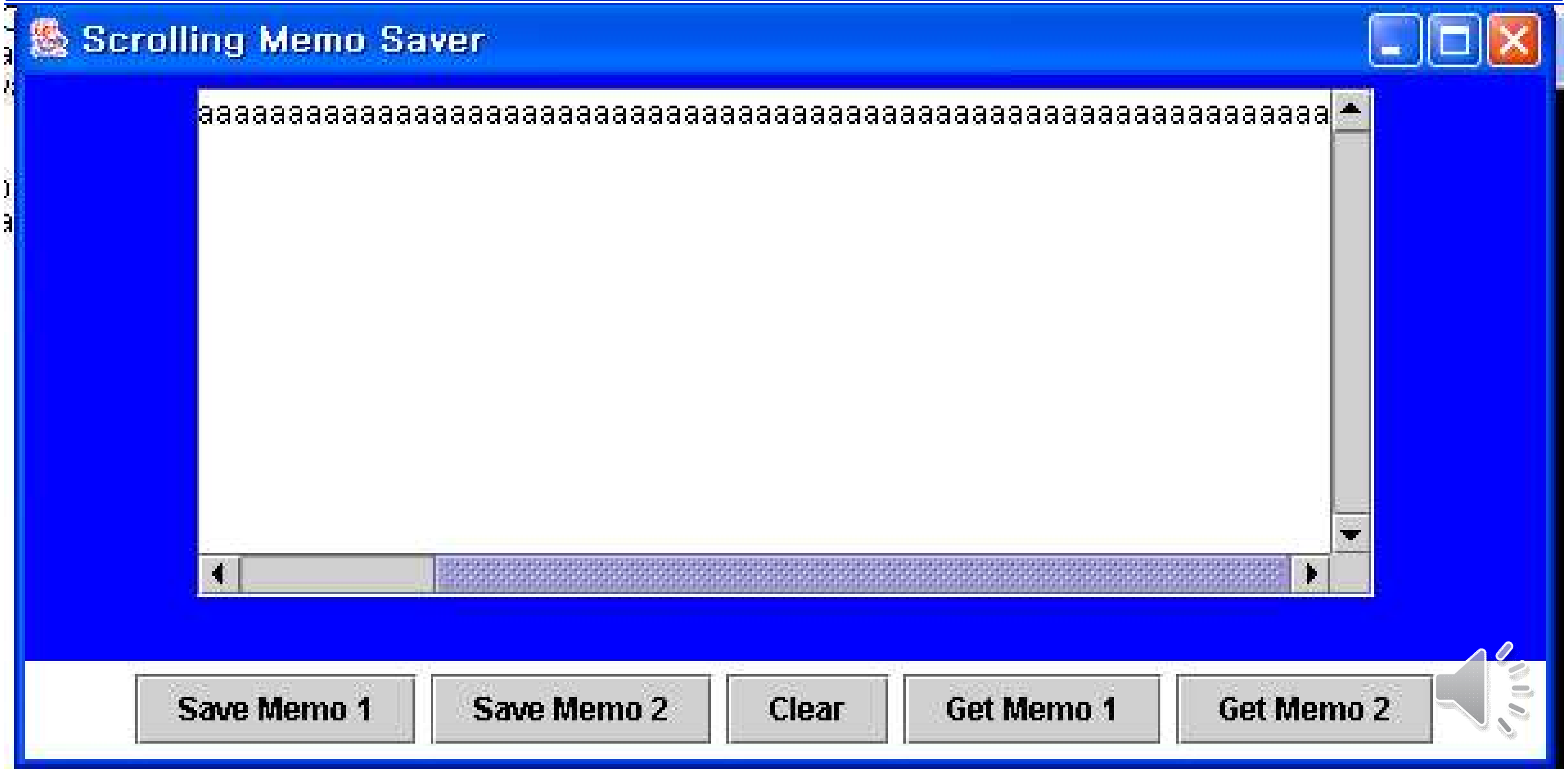
```
 JButton get2Button = new JButton("Get Memo 2");
        get2Button.addActionListener(this);
        buttonPanel.add(get2Button);
        contentPane.add(buttonPanel, BorderLayout.SOUTH);

        JPanel textPanel = new JPanel( );
        textPanel.setBackground(Color.BLUE);
        theText = new JTextArea(LINES, CHAR_PER_LINE);
        theText.setBackground(Color.WHITE);
        //
        JScrollPane scrolledText = new JScrollPane(theText);
        //scrolledText.setHorizontalScrollBarPolicy(
        //              JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        scrolledText.setVerticalScrollBarPolicy(
                    JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
        textPanel.add(scrolledText);
        contentPane.add(textPanel, BorderLayout.CENTER);
        // The omitted statements are the same as in Display 14.1.
        // in fact, this constructor is the same as that in Display 14.1,
        // except that the name of the class and the title of the JFrame are
changed
        // and the forgoing four highlighted statement replace a single
statement
        // in Display 14.1. All other methods are the same as in Display 14.1
    }
```

```
public void actionPerformed(ActionEvent e)
   {
      String actionCommand = e.getActionCommand( );
      if (actionCommand.equals("Save Memo 1"))
         memo1 = theText.getText( );
      else if (actionCommand.equals("Save Memo 2"))
         memo2 = theText.getText( );
      else if (actionCommand.equals("Clear"))
         theText.setText("");
      else if (actionCommand.equals("Get Memo 1"))
         theText.setText(memo1);
      else if (actionCommand.equals("Get Memo 2"))
         theText.setText(memo2);
      else
         theText.setText("Error in memo interface");
   }

   public static void main(String[] args)
   {
      ScrollBarDemo guiMemo = new ScrollBarDemo( );
      guiMemo.setVisible(true);
   }
}
```


Scrolling Memo Saver
Save Memo 1
Save Memo 2
Clear
Get Memo 1
Get Memo 2

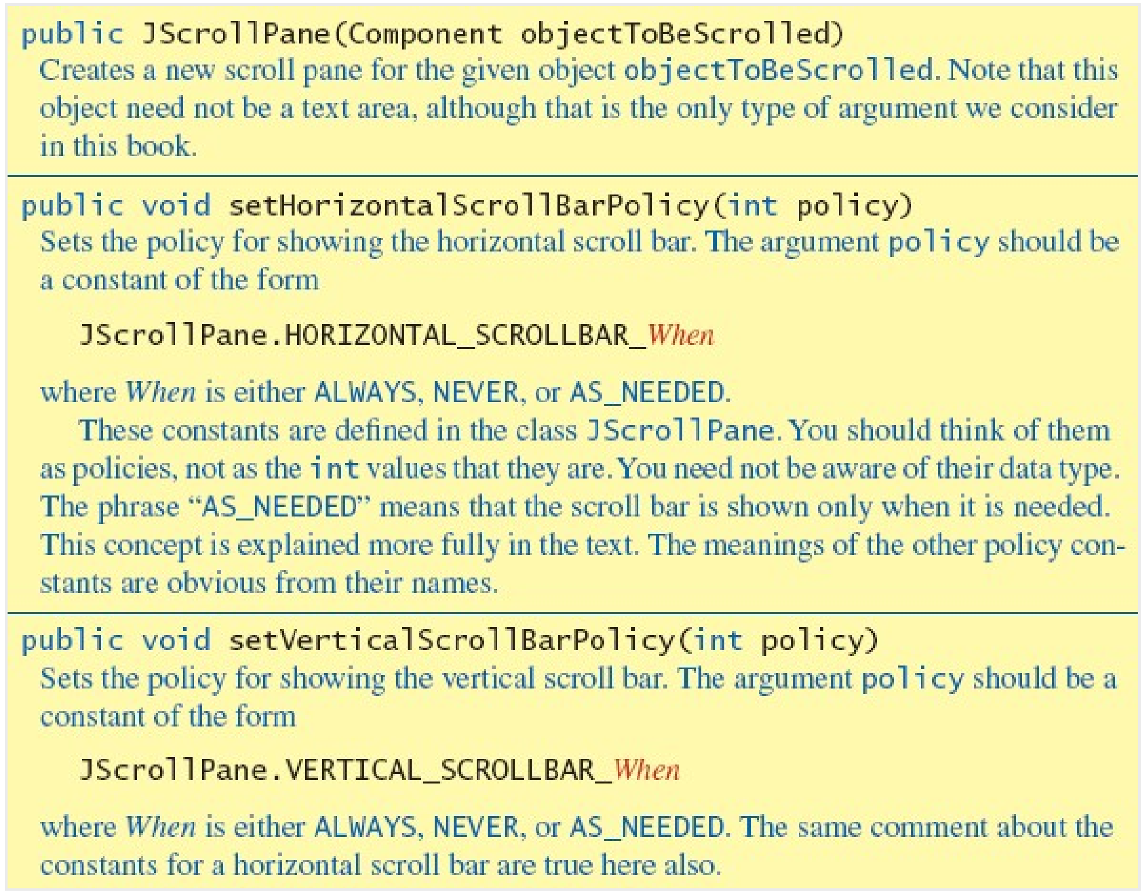
```
g
g
g

g
g
g
g
g
g
```

```
JScrollPane scrolledText = new JScrollPane(theText);
scrolledText.setHorizontalScrollBarPolicy(
        JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
scrolledText.setVerticalScrollBarPolicy(
        JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
textPanel.add(scrolledText);
contentPane.add(textPanel, BorderLayout.CENTER);
```



JScrollPane methods

- Figure 15.3 Some methods and constants in **`class JScrolPane`**



`public JScrollPane(Component objectToBeScrolled)`

Creates a new scroll pane for the given object `objectToBeScrolled`. Note that this object need not be a text area, although that is the only type of argument we consider in this book.

`public void setHorizontalScrollBarPolicy(int policy)`

Sets the policy for showing the horizontal scroll bar. The argument `policy` should be a constant of the form

`JScrollPane.HORIZONTAL_SCROLLBAR_`*When*

where *When* is either `ALWAYS`, `NEVER`, or `AS_NEEDED`.

These constants are defined in the class `JScrollPane`. You should think of them as policies, not as the `int` values that they are. You need not be aware of their data type. The phrase "`AS_NEEDED`" means that the scroll bar is shown only when it is needed. This concept is explained more fully in the text. The meanings of the other policy constants are obvious from their names.

`public void setVerticalScrollBarPolicy(int policy)`

Sets the policy for showing the vertical scroll bar. The argument `policy` should be a constant of the form

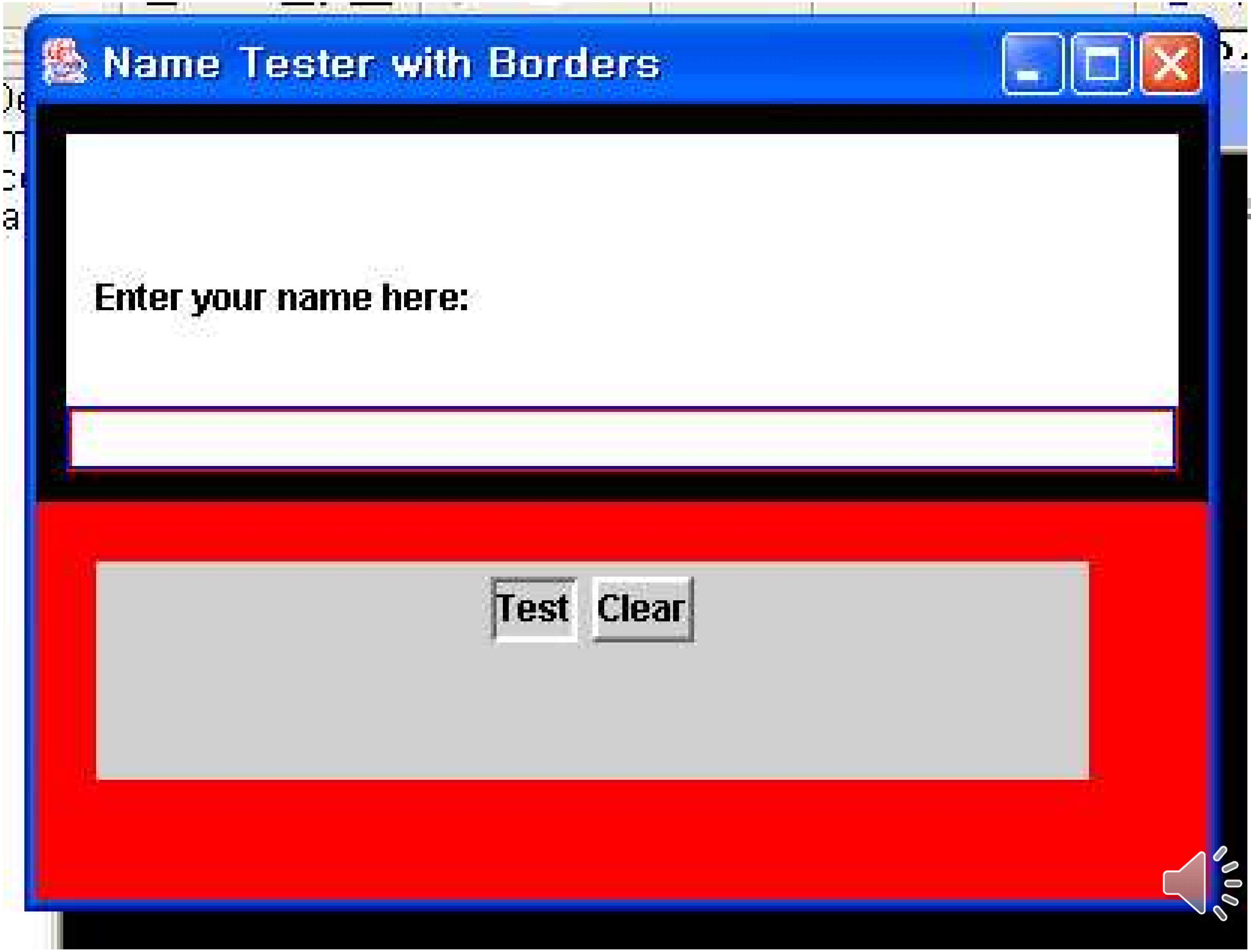
`JScrollPane.VERTICAL_SCROLLBAR_`*When*

where *When* is either `ALWAYS`, `NEVER`, or `AS_NEEDED`. The same comment about the constants for a horizontal scroll bar are true here also.

Adding Borders

- A *border* is an area that frames a component.
- Swing provides several different types of borders:
  - » `BevelBorder`—makes component look raised or lowered
    - – BevelBorder.RAISED, BevelBorder.LOWERED
  - » `EtchedBorder`—similar to `BevelBorder` but can't set size
    - – `highlight, shadow`
  - » `EmptyBorder`—extra space around the component
  - » `LineBorder`—colored border of a given thickness
  - » `MatteBorder`—similar to `LineBorder` but can adjust thickness on each side of the component
- An example of adding a bevel border to a button:

```
testButton.setBorder(new BevelBorder(BevelBorder.LOWERED));
```


Name Tester with Borders
Enter your name here:
Test
Clear

Some Border Classes

- to create a `BevelBorder` object

  ```
  public BevelBorder(int bevelType)
  ```

  where `bevelType` is one of

  ```
  BevelBorder.RAISED
  BevelBorder.LOWERED
  ```

Some Border Classes, cont.

- to create an `EtchedBorder` object

```
public  EtchedBorder(int etchType,
     Color highlight, Color shadow)
```

  where `etchType` is one of

```
EtchedBorder.RAISED
EtchedBorder.LOWERE
```

- to create an `EtchedBorder` object

```
public EtchedBorder(Color highlight,
     Color shadow)
```

  » Creates a lowered etched border with the specified highlight and shadow colors.

  » **Parameters:**

    – highlight - the color to use for the etched highlight

    – shadow - the color to use for the etched shadow

Some Border Classes, cont.

- to create an `EmptyBorder` object

```
public EmptyBorder(int top, int left,
     int bottom, int right)
```

- to create an `LineBorder` object

```
public LIneBorderBorder(Color theColor,
     int thickness)
```

- to create an `LineBorder` object

```
public LineBorderBorder(Color theColor,
     int thickness,
     boolean roundedCorners)
```

Some Border Classes, cont.

- to create an `MatteBorder` object

```
public matteBorder(int top, int left,
     int bottom, int right,
     Color theColor)
```

- to create an `MatteBorder` object with an icon

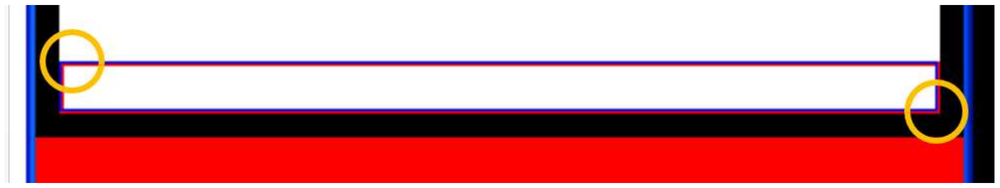
```
public MatteBorder(int top, int left,
     int bottom, int right,
     ImageIcon theIcon)
```

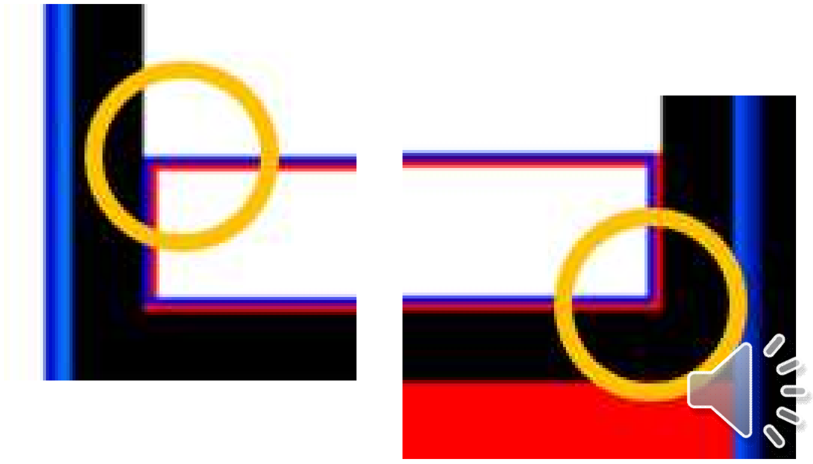
Listing 15.4 Demonstration of Different Borders - BorderDemo.java

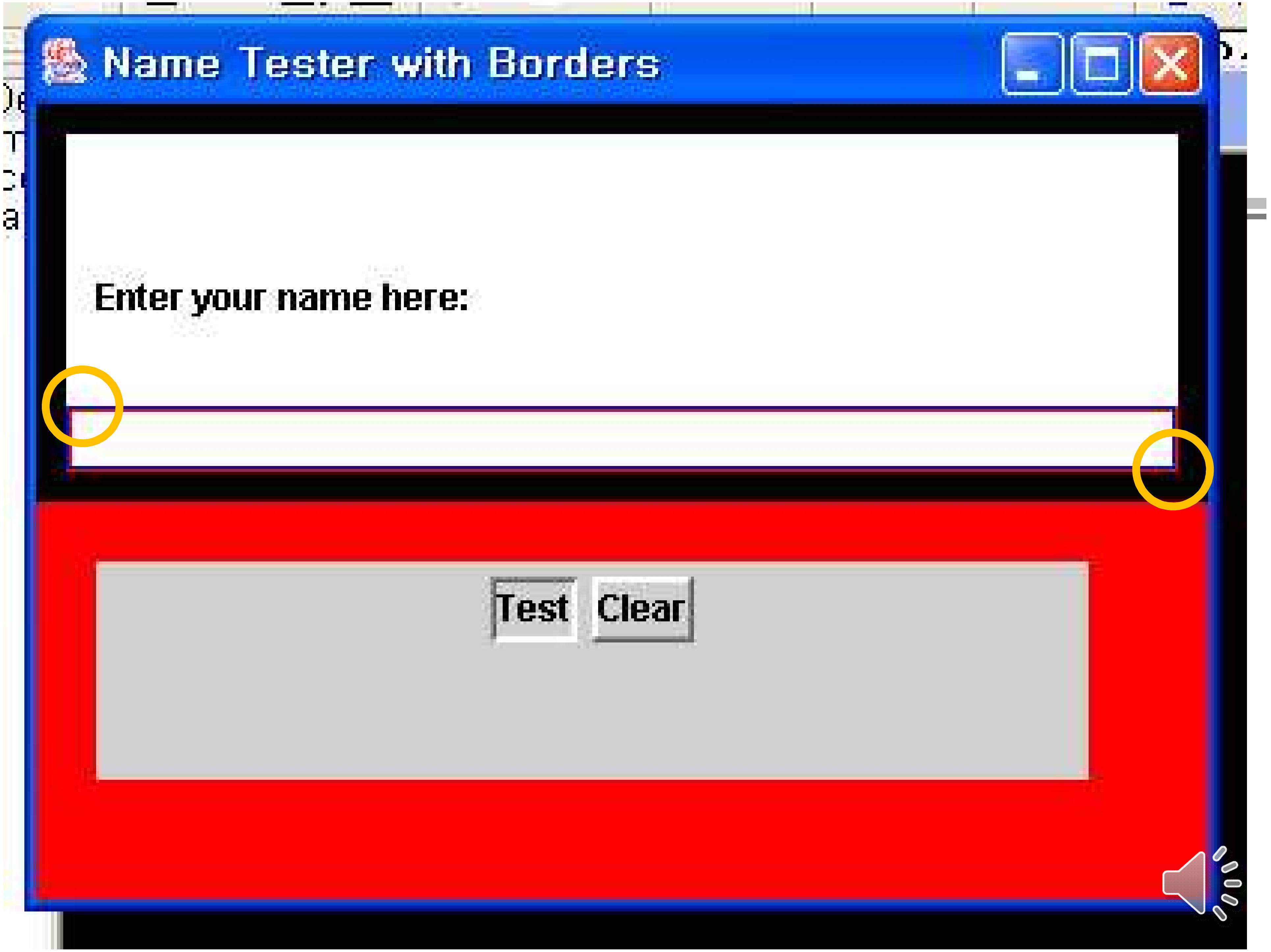
- import javax.swing.border.*;

```
// Listing 15.4  Demonstration of Different Borders

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;

/**
 Class to demonstrate adding borders to components.
*/
public class BorderDemo extends JFrame implements ActionListener
{
   public static final int WIDTH = 400;
   public static final int HEIGHT = 300;

   private JTextField name;
```

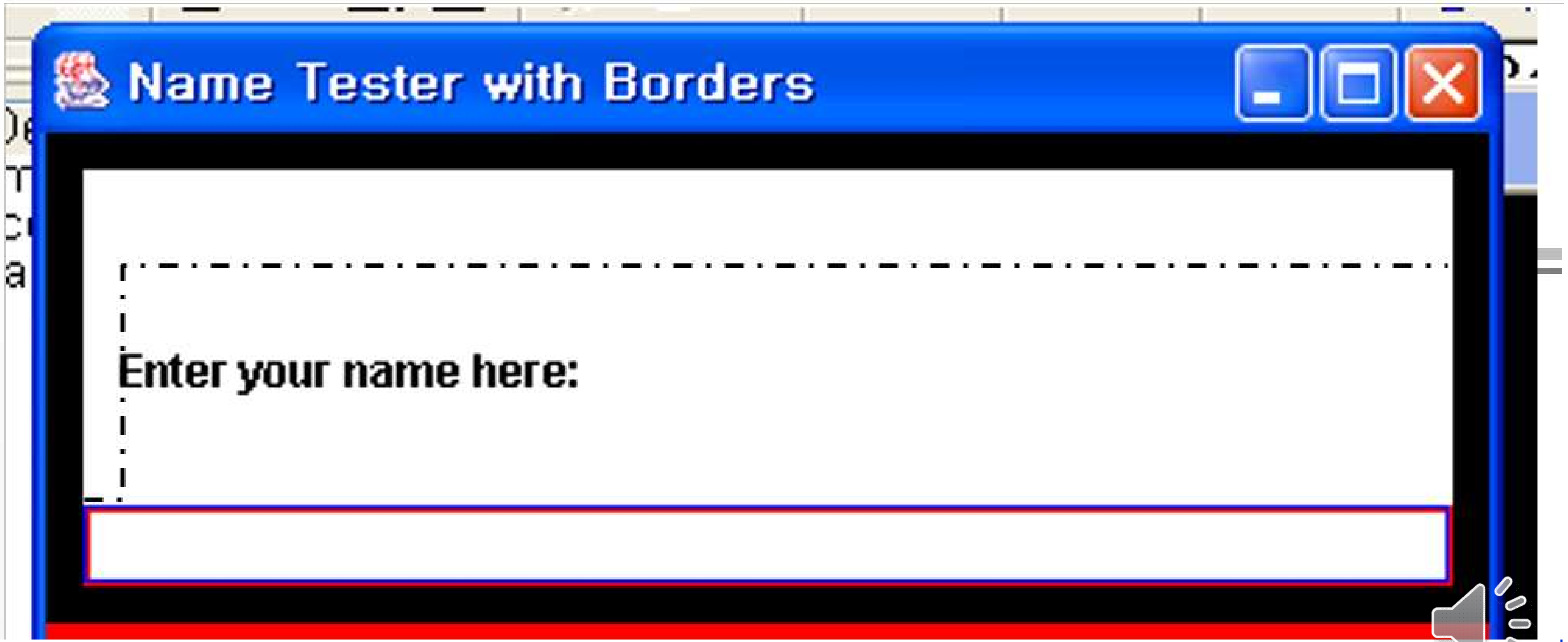
```
public BorderDemo( )
  {
     setTitle("Name Tester with Borders");
     setSize(WIDTH, HEIGHT);
     addWindowListener(new WindowDestroyer( ));
     Container content = getContentPane( );
     content.setLayout(new GridLayout(2, 1));

     JPanel namePanel = new JPanel( );
     namePanel.setLayout(new BorderLayout( ));
     namePanel.setBackground(Color.WHITE);

     name = new JTextField(20);
     //The following border is not as dramatic as others,
     //but look closely and you will see it.
     // name.setBorder(new EtchedBorder(Color.GREEN, Color.BLUE));
     name.setBorder(new EtchedBorder(Color.RED, Color.BLUE));
```

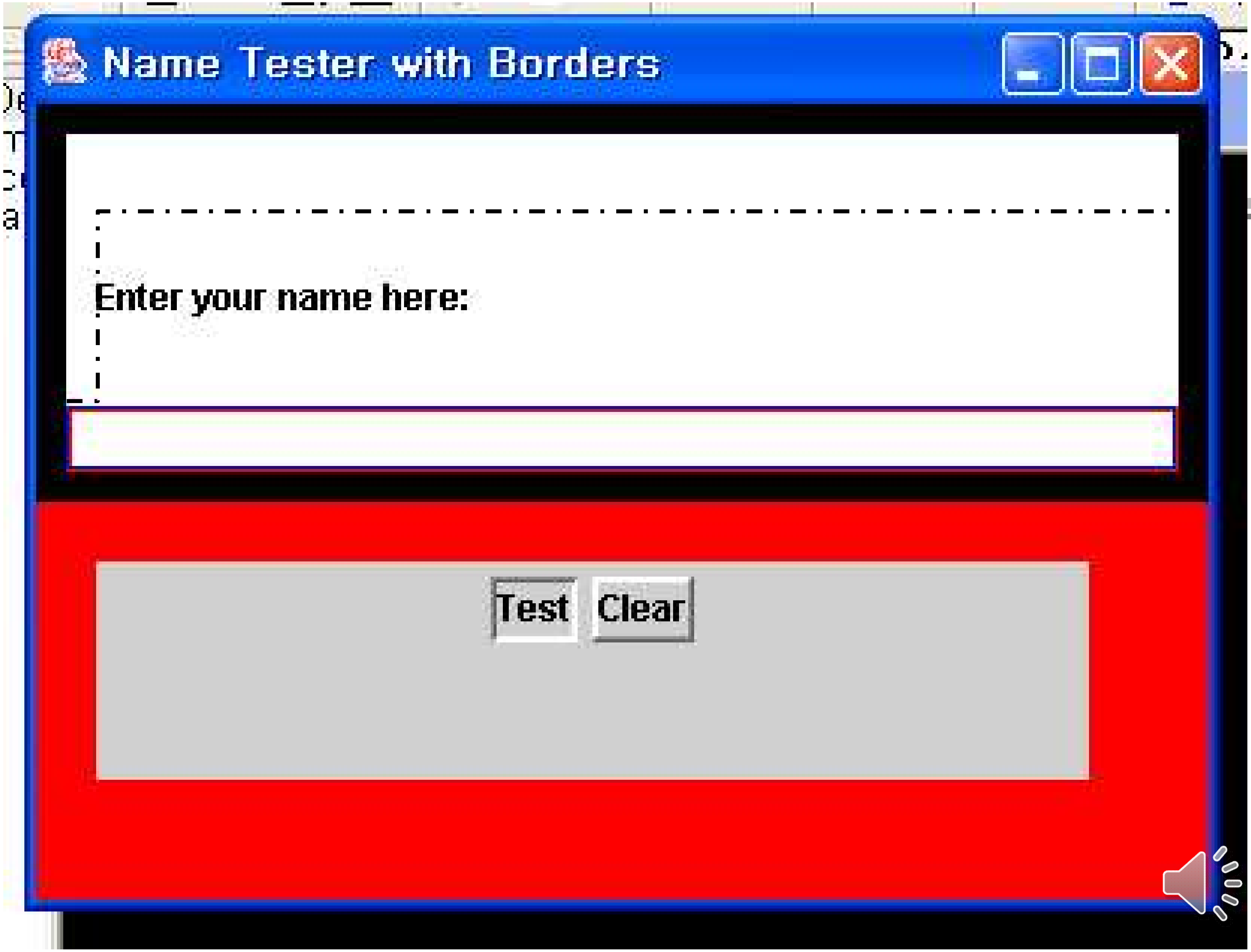




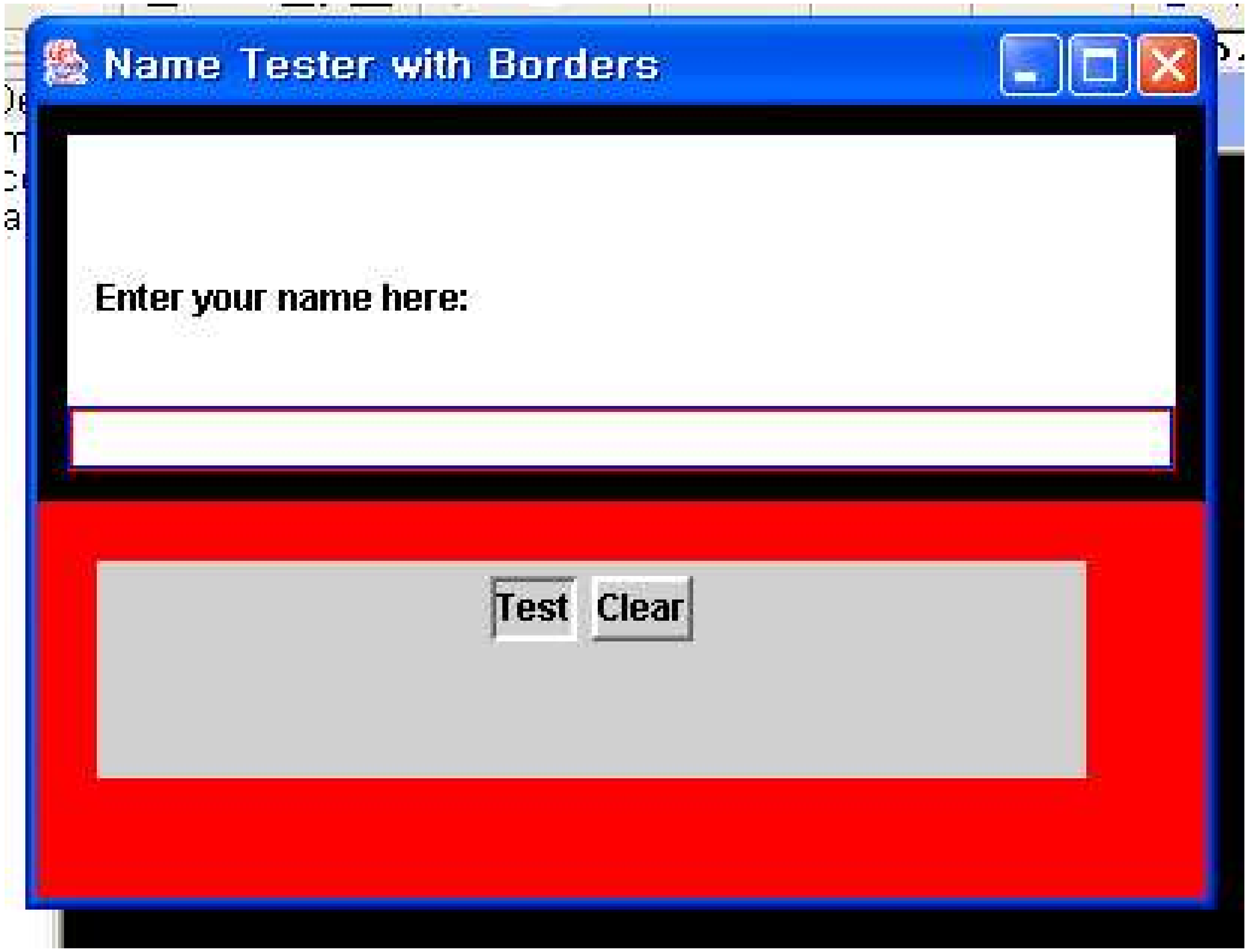

Name Tester with Borders
Enter your name here:
Test
Clear

```
namePanel.add(name, BorderLayout.SOUTH);
JLabel nameLabel = new JLabel("Enter your name here:");
//The following does insert space around the label.
//To see the difference, comment out the following line:
nameLabel.setBorder(new EmptyBorder(20, 10, 0, 0));
                                    // top, left, bottom, right
namePanel.add(nameLabel, BorderLayout.CENTER);
```

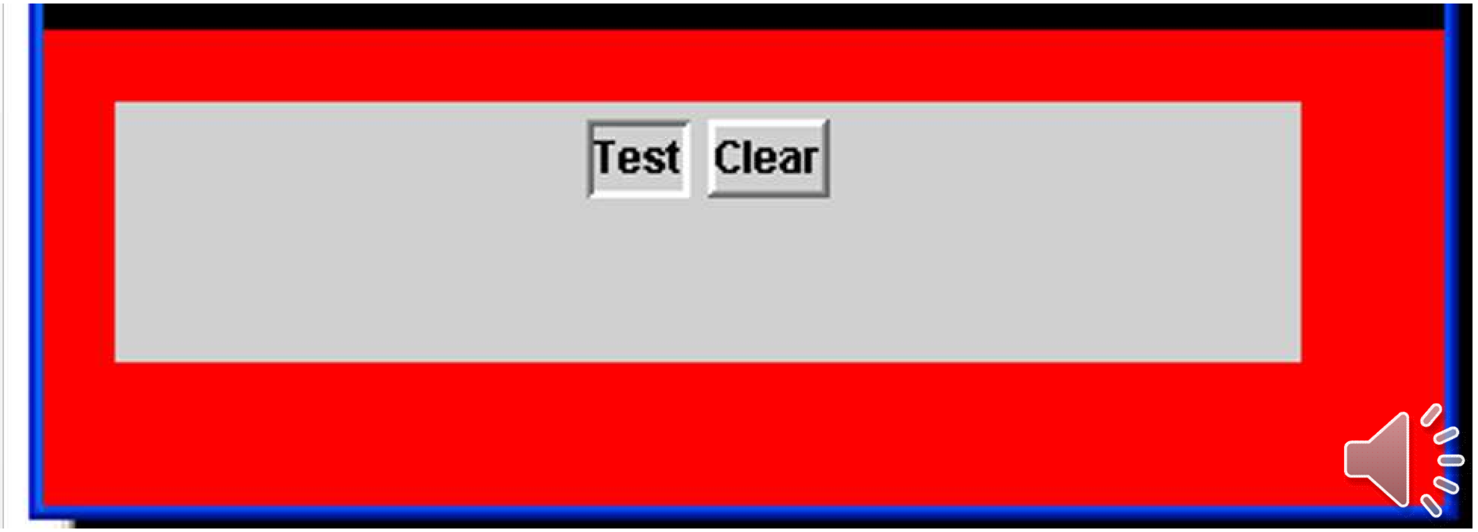



Name Tester with Borders
Enter your name here:
Test
Clear

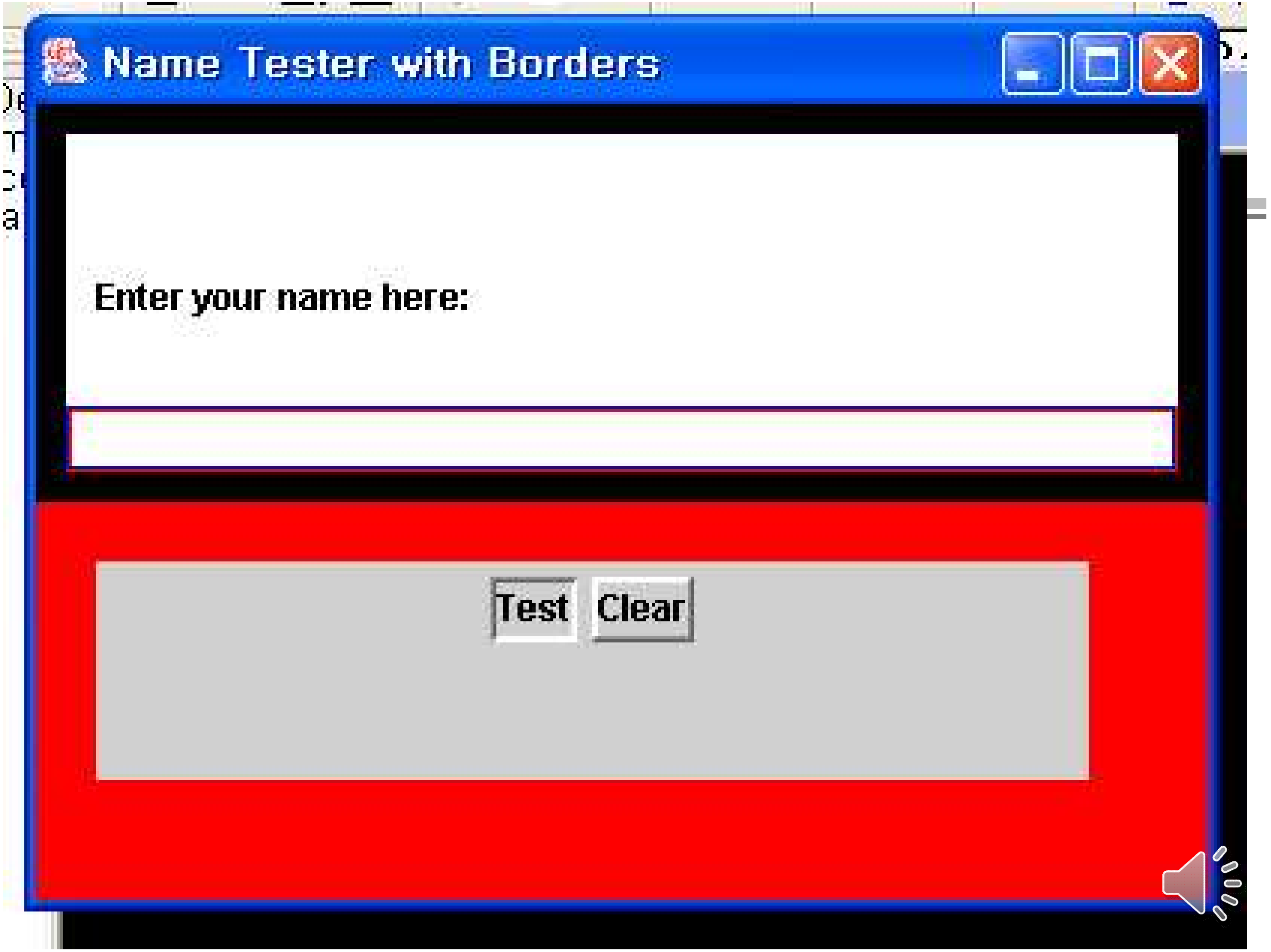
```
 namePanel.setBorder(new LineBorder(Color.BLACK, 10));
 //       namePanel.setBorder(new LineBorder(Color.RED, 5));
content.add(namePanel);
```



```
    JPanel buttonPanel = new JPanel( );
    buttonPanel.setLayout(new FlowLayout( ));
    JButton testButton = new JButton("Test");
    testButton.addActionListener(this);
    testButton.setBorder(new BevelBorder(BevelBorder.LOWERED));
    buttonPanel.add(testButton);

    JButton clearButton = new JButton("Clear");
    clearButton.addActionListener(this);
    clearButton.setBorder(new BevelBorder(BevelBorder.RAISED));
    buttonPanel.add(clearButton);

    buttonPanel.setBorder(
//        new MatteBorder(60, 40, 30, 20, Color.PINK));
        new MatteBorder(20, 20, 40, 40, Color.RED));
    content.add(buttonPanel);
}
```

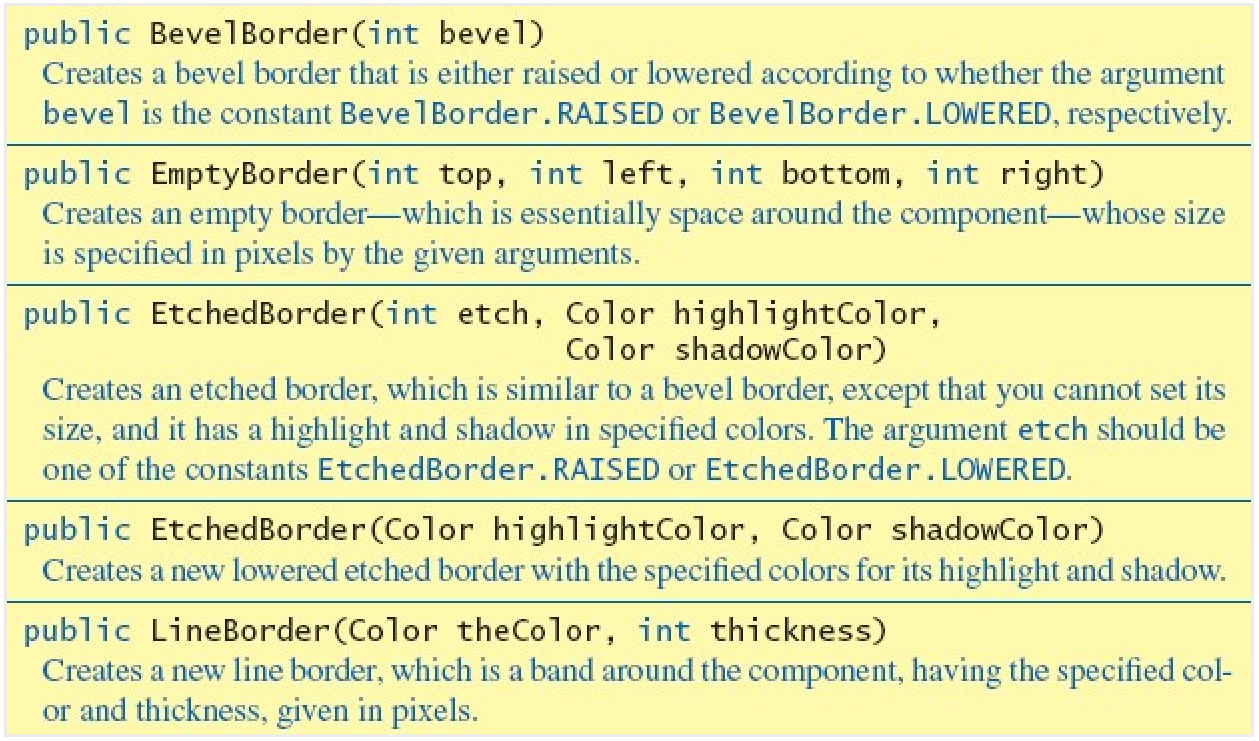


```
 public void actionPerformed(ActionEvent e)
   {
       if (e.getActionCommand( ).equals("Test"))
           name.setText("A very good name!");
       else if (e.getActionCommand( ).equals("Clear"))
           name.setText("");
       else
           name.setText("Error in window interface.");
   }

   public static void main(String[] args)
   {
       BorderDemo w = new BorderDemo( );
       w.setVisible(true);
   }
}
```


Name Tester with Borders
Enter your name here:
Test
Clear

Border Class Constructors

- Figure 15.4a, some constructors of various border classes



```
public BevelBorder(int bevel)
```
Creates a bevel border that is either raised or lowered according to whether the argument `bevel` is the constant `BevelBorder.RAISED` or `BevelBorder.LOWERED`, respectively.

```
public EmptyBorder(int top, int left, int bottom, int right)
```
Creates an empty border—which is essentially space around the component—whose size is specified in pixels by the given arguments.

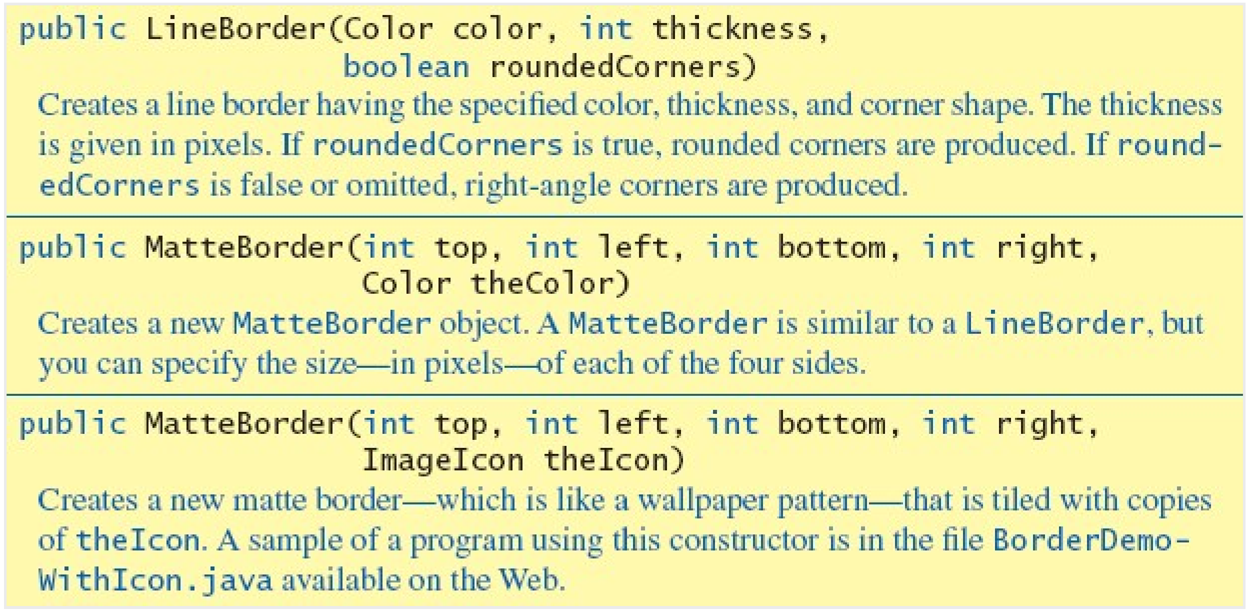
```
public EtchedBorder(int etch, Color highlightColor,
                              Color shadowColor)
```
Creates an etched border, which is similar to a bevel border, except that you cannot set its size, and it has a highlight and shadow in specified colors. The argument `etch` should be one of the constants `EtchedBorder.RAISED` or `EtchedBorder.LOWERED`.

```
public EtchedBorder(Color highlightColor, Color shadowColor)
```
Creates a new lowered etched border with the specified colors for its highlight and shadow.

```
public LineBorder(Color theColor, int thickness)
```
Creates a new line border, which is a band around the component, having the specified color and thickness, given in pixels.

Border Class Constructors

- Figure 15.4b, some constructors of various border classes



```
public LineBorder(Color color, int thickness,
                  boolean roundedCorners)
```

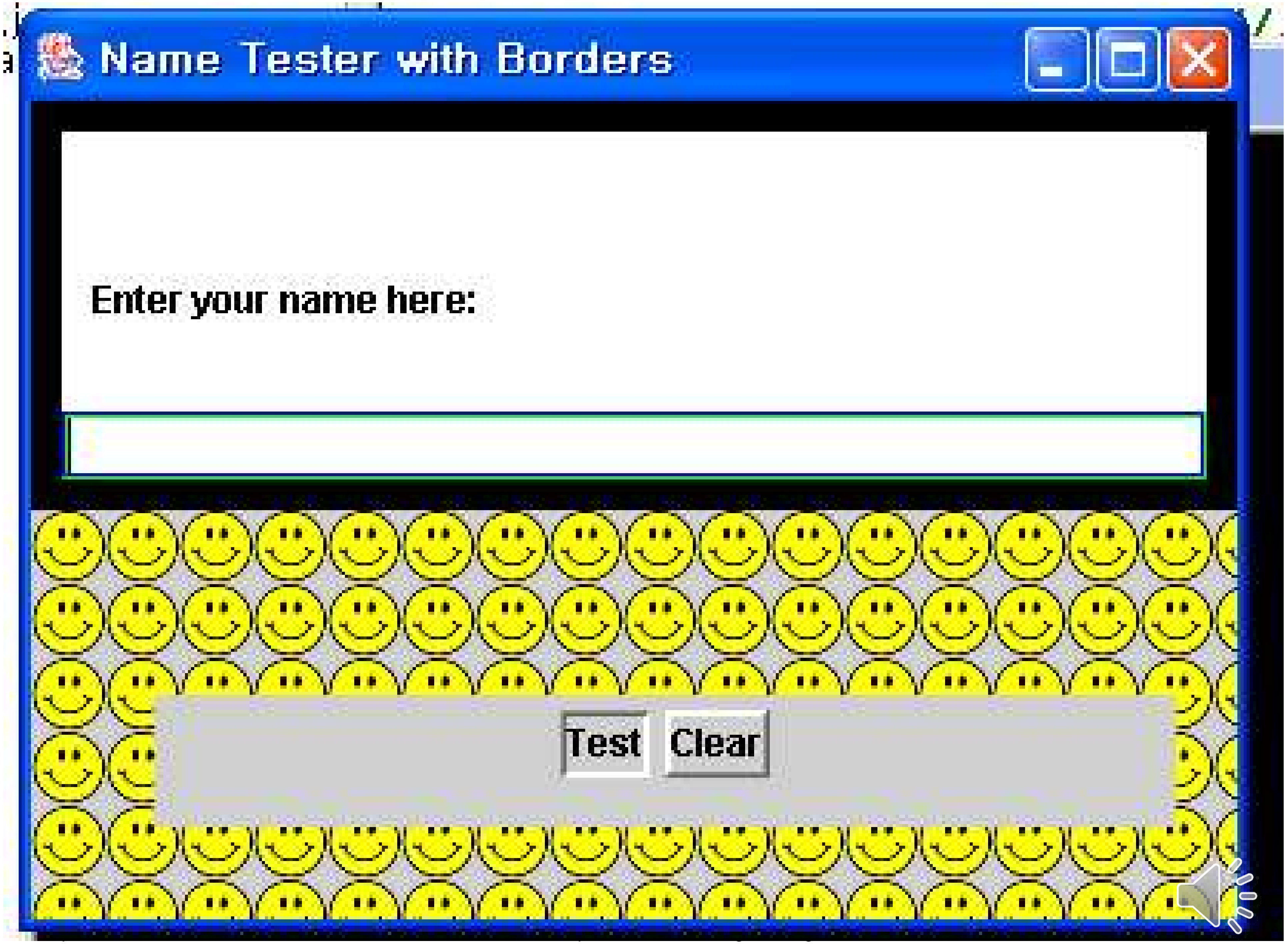
Creates a line border having the specified color, thickness, and corner shape. The thickness is given in pixels. If `roundedCorners` is true, rounded corners are produced. If `round-edCorners` is false or omitted, right-angle corners are produced.

```
public MatteBorder(int top, int left, int bottom, int right,
                   Color theColor)
```

Creates a new `MatteBorder` object. A `MatteBorder` is similar to a `LineBorder`, but you can specify the size—in pixels—of each of the four sides.

```
public MatteBorder(int top, int left, int bottom, int right,
                   ImageIcon theIcon)
```

Creates a new matte border—which is like a wallpaper pattern—that is tiled with copies of `theIcon`. A sample of a program using this constructor is in the file `BorderDemo-WithIcon.java` available on the Web.

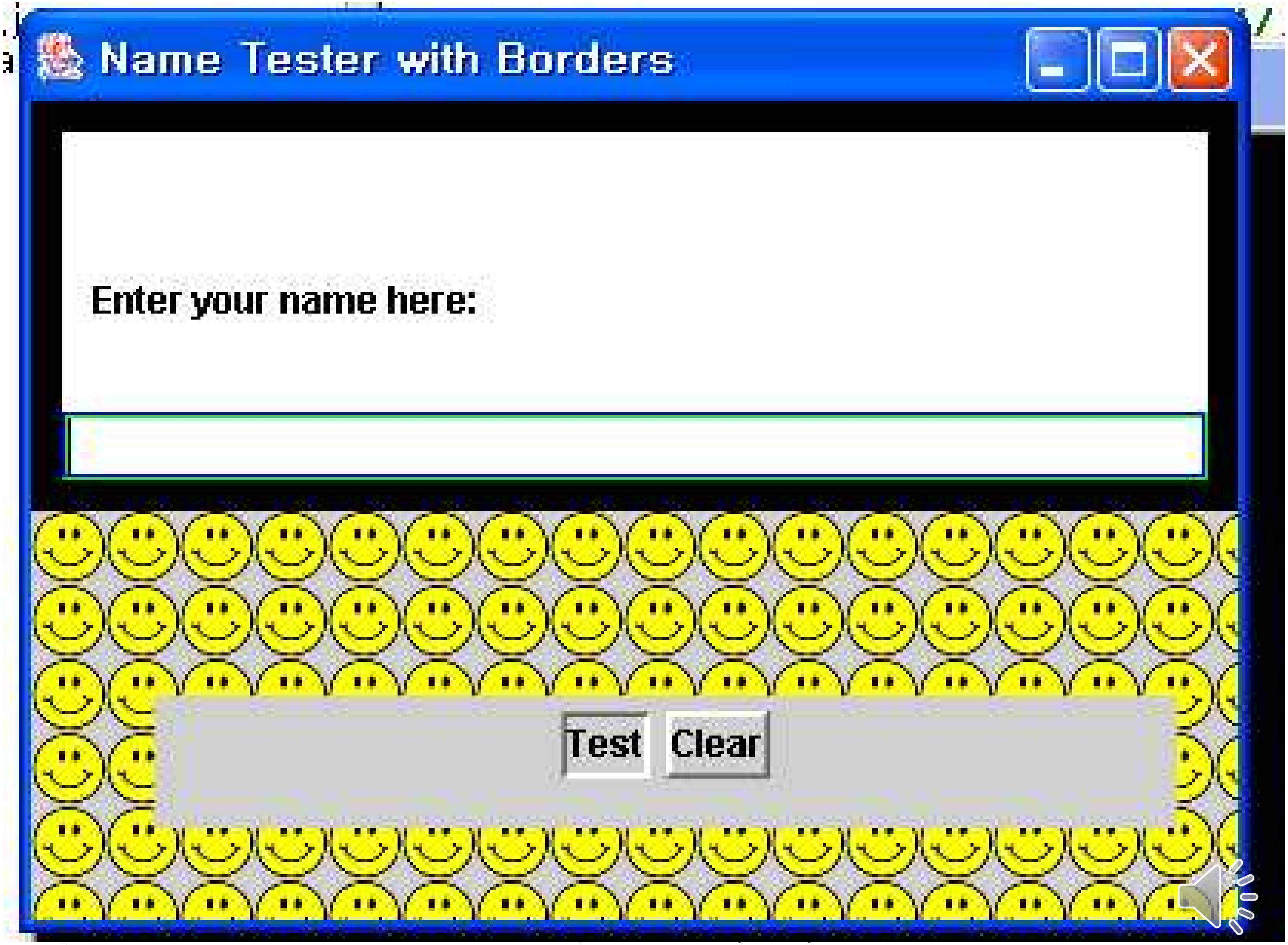

Name Tester with Borders
Enter your name here:
Test
Clear

Extra Code on CD BorderDemoWithIcon.Java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;

/**
 Class to demonstrate adding icons to a border.
*/
public class BorderDemoWithIcon extends JFrame implements
ActionListener
{
    public static final int WIDTH = 400;
    public static final int HEIGHT = 300;

    private JTextField name;
```

```
public BorderDemoWithIcon()
  {
     setTitle("Name Tester with Borders");
     setSize(WIDTH, HEIGHT);
     addWindowListener(new WindowDestroyer());
     Container content = getContentPane();
     content.setLayout(new GridLayout(2, 1));

     JPanel namePanel = new JPanel();
     namePanel.setLayout(new BorderLayout());
     namePanel.setBackground(Color.WHITE);

     name = new JTextField(20);
     //The following border is not as dramatic as others,
     //but look closely and you will see it.
     name.setBorder(new EtchedBorder(Color.GREEN, Color.BLUE));
     namePanel.add(name, BorderLayout.SOUTH);
     JLabel nameLabel = new JLabel("Enter your name here:");
     //The following does insert space around the label.
     //To see the difference, comment out the following line:
     nameLabel.setBorder(new EmptyBorder(20, 10, 0, 0));
     namePanel.add(nameLabel, BorderLayout.CENTER);
```

```
 namePanel.setBorder(new LineBorder(Color.BLACK, 10));
        content.add(namePanel);

        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new FlowLayout());
        JButton testButton = new JButton("Test");
        testButton.addActionListener(this);
        testButton.setBorder(new BevelBorder(BevelBorder.LOWERED));
        buttonPanel.add(testButton);

        JButton clearButton = new JButton("Clear");
        clearButton.addActionListener(this);
        clearButton.setBorder(new BevelBorder(BevelBorder.RAISED));
        buttonPanel.add(clearButton);

////////////////////////////////////////////////////////////////////
        ImageIcon smileyIcon = new ImageIcon("smiley.gif");
        buttonPanel.setBorder(
                new MatteBorder(60, 40, 30, 20, smileyIcon));
        content.add(buttonPanel);
    }
 public void actionPerformed(ActionEvent e)
   {
       if (e.getActionCommand().equals("Test"))
           name.setText("A very good name!");
       else if (e.getActionCommand().equals("Clear"))
           name.setText("");
       else
           name.setText("Error in window interface.");
   }

   public static void main(String[] args)
   {
       BorderDemoWithIcon w = new BorderDemoWithIcon();
       w.setVisible(true);
   }
}
```


Name Tester with Borders
Enter your name here:
Test
Clear

끝