

CHAPTER

10

GRAPHICAL USER INTERFACES





Chapter Goals

- ❑ To implement simple graphical user interfaces
- ❑ To add buttons, text fields, and other components to a frame window
- ❑ To handle events that are generated by buttons
- ❑ To write programs that display simple drawings

In this chapter, you will learn how to write graphical user-interface applications, process the events that are generated by button clicks, and process user input,



Contents

- ❑ Frame Windows
- ❑ Events and Event Handling
- ❑ Processing Text Input
- ❑ Creating Drawings



10.1 Frame Windows

- Java provides classes to create graphical applications that can run on any major graphical user interface
 - A graphical application shows information inside a frame: a window with a title bar
- Java's `JFrame` class allows you to display a frame
 - It is part of the `javax.swing` package





The JFrame Class

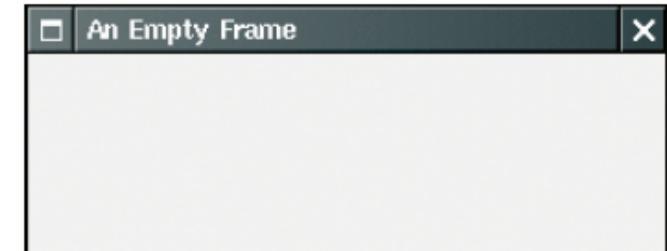
- Five steps to displaying a frame:

- 1) Construct an object of the JFrame class

```
JFrame frame = new JFrame();
```

- 2) Set the size of the frame

```
frame.setSize(300,400);
```



- 3) Set the title of the frame

```
frame.setTitle("An Empty Frame");
```

- 4) Set the “default close operation”

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

- 5) Make it visible

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



EmptyFrameViewer.java

- Your JVM (Java Virtual Machine) does all of the work of displaying the frame on your GUI
 - This application is portable to all supported GUIs!

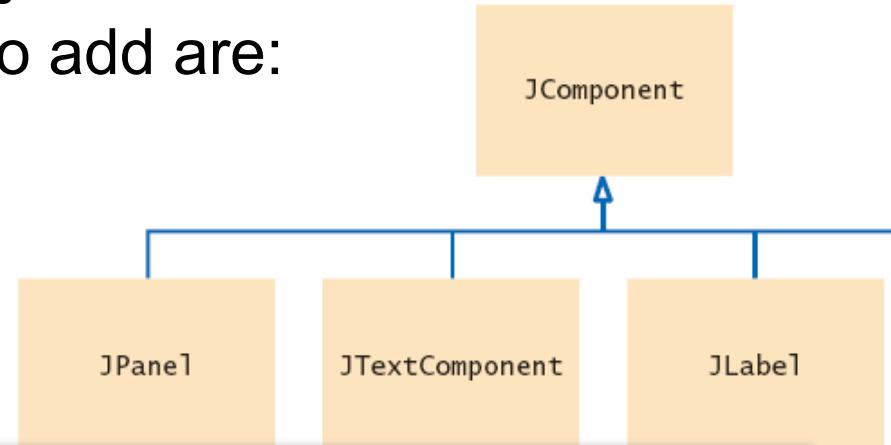
```
1 import javax.swing.JFrame;
2 /**
3  * This program displays an empty frame.
4 */
5 public class EmptyFrameViewer
6 {
7     public static void main(String[] args)
8     {
9         JFrame frame = new JFrame();
10
11         final int FRAME_WIDTH = 300;
12         final int FRAME_HEIGHT = 400;
13         frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
14         frame.setTitle("An empty frame");
15         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16
17         frame.setVisible(true);
18     }
19 }
20 }
```

You are using the java Swing library



Adding Components

- You cannot draw directly on a JFrame object
- Instead, construct an object and add it to the frame
 - A few examples objects to add are:
 - JComponent
 - JPanel
 - JTextComponent
 - JLabel



```
public class RectangleComponent extends JComponent
{
    public void paintComponent(Graphics g)
    {
        // Drawing instructions go here
    }
}
```

Extend the JComponent Class and override its paintComponent method



Adding Panels

- ❑ If you have more than one component, put them into a panel (a container for other user-interface components), and then add the panel to the frame:

- First Create the components

```
JButton button = new JButton("Click me!");  
JLabel label = new JLabel("Hello, World!");
```

- Then Add them to the panel

```
JPanel panel = new JPanel();  
panel.add(button);  
panel.add(label);  
frame.add(panel);
```

Use a JPanel to group multiple user-interface components together.

Add the panel to the frame



FilledFrameViewer.java

```
1 import javax.swing.JButton;
2 import javax.swing.JFrame;
3 import javax.swing.JLabel;
4 import javax.swing.JPanel;
5
6 /**
7  * This program shows a frame that is filled with two components.
8 */
9 public class FilledFrameViewer
10 {
11     public static void main(String[] args)
12     {
13         JFrame frame = new JFrame();
14
15         JButton button = new JButton("Click me!");
16         JLabel label = new JLabel("Hello, World!");
17
18         JPanel panel = new JPanel();
19         panel.add(button);
20         panel.add(label);
21         frame.add(panel);
22
23         final int FRAME_WIDTH = 300;
24         final int FRAME_HEIGHT = 100;
25         frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
26         frame.setTitle("A frame with two components");
27         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
28
29         frame.setVisible(true);
30     }
31 }
```



Using Inheritance to Customize Frames

For complex frames:

- Design a subclass of JFrame
- Store the components as instance variables
- Initialize them in the constructor of your subclass.

```
public class FilledFrame extends JFrame  
{  
    private JButton button;      Components are instance variables  
    private JLabel label;  
    private static final int FRAME_WIDTH = 300;  
    private static final int FRAME_HEIGHT = 100;  
  
    public FilledFrame()  
    {  
        createComponents();          Initialize and add them in the  
        setSize(FRAME_WIDTH, FRAME_HEIGHT);  constructor of your subclass with  
    }                                a helper method  
}
```



Using Inheritance to Customize Frames

- Then instantiate the customized frame from the main method

```
public class FilledFrameViewer2
{
    public static void main(String[] args)
    {
        JFrame frame = new FilledFrame();
        frame.setTitle("A frame with two components");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```



Special Topic 10.1



- Adding the main Method to the Frame Class
 - Some programmers prefer this technique

```
public class FilledFrame extends JFrame  
{  
    . . .  
    public static void main(String[] args)  
    {  
        JFrame frame = new FilledFrame();  
        frame.setTitle("A frame with two components");  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        frame.setVisible(true);  
    }  
    public FilledFrame()  
    {  
        createComponents();  
        setSize(FRAME_WIDTH, FRAME_HEIGHT);  
    }  
    . . .  
}
```

Once main has instantiated the FilledFrame, non-static instance variables and methods can be used.



10.2 Events and Event Handling

- In a modern **graphical user interface** program, the user controls the program through the mouse and keyboard.
- The user can enter information into text fields, pull down menus, click buttons, and drag scroll bars in any order.
 - The program must react to the user commands
 - The program can choose to receive and handle events such as “mouse move” or a button push “action event”



Events and Action Listeners

- ❑ Programs must indicate which events it wants to receive
- ❑ It does so by installing event listener objects
 - An event listener object belongs to a class that you declare
 - The methods of your event listener classes contain the instructions that you want to have executed when the events occur
- ❑ To install a listener, you need to know the **event source**
- ❑ You add an event listener object to selected event sources:
 - Examples: OK Button clicked, Cancel Button clicked, Menu Choice..
- ❑ Whenever the event occurs, the event source calls the appropriate methods of all attached event listeners



Example ActionListener

- The ActionListener interface has one method:

```
public interface ActionListener
{
    void actionPerformed(ActionEvent event);
}
```

- ClickListener class implements the ActionListener interface

```
1 import java.awt.event.ActionEvent;
2 import java.awt.event.ActionListener;
3
4 /**
5     An action listener that prints a message.
6 */
7 public class ClickListener implements ActionListener
8 {
9     public void actionPerformed(ActionEvent event)
10    {
11        System.out.println("I was clicked.");
12    }
13 }
```

We can ignore the event parameter – it has information such as when the event occurred



Registering ActionListener

- A ClickListener object must be created, and then ‘registered’ (added) to a specific event source

```
ActionListener listener = new ClickListener();
button.addActionListener(listener);
```

- Now whenever the button object is clicked, it will call `listener.ActionPerformed`, passing it the event as a parameter

```
Terminal
~/books/bjol/code/ch10/sec2_1$ I was clicked.
I was clicked.
I was clicked.
```



```
1 import java.awt.event.ActionEvent;
2 import java.awt.event.ActionListener;
3
4 /**
5      An action listener that prints a message.
6 */
7 public class ClickListener implements ActionListener
8 {
9     public void actionPerformed(ActionEvent event)
10    {
11        System.out.println("I was clicked.");
12    }
13 }
```



ButtonFrame1.java

```
6  /**
7   * This frame demonstrates how to install an action listener.
8  */
9 public class ButtonFrame1 extends JFrame
10 {
11     private static final int FRAME_WIDTH = 100;
12     private static final int FRAME_HEIGHT = 60;
13
14     public ButtonFrame1()
15     {
16         createComponents();
17         setSize(FRAME_WIDTH, FRAME_HEIGHT);
18     }
19
20     private void createComponents()
21     {
22         JButton button = new JButton("Click me!");
23         JPanel panel = new JPanel();
24         panel.add(button);
25         add(panel);
26
27         ActionListener listener = new ClickListener();
28         button.addActionListener(listener);
29     }
30 }
```

Creates and adds a JButton to the frame

Tells the button to ‘call us back’ when an event occurs.



ButtonViewer1.java

- ❑ No changes required to the main to implement an event handler

```
1 import javax.swing.JFrame;
2
3 /**
4     This program demonstrates how to install an action listener.
5 */
6 public class ButtonViewer1
7 {
8     public static void main(String[] args)
9     {
10         JFrame frame = new ButtonFrame1();
11         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12         frame.setVisible(true);
13     }
14 }
```



Inner Classes for Listeners

- ❑ In the preceding section, you saw how the code that is executed when a button is clicked is placed into a listener class.
- ❑ Inner Classes are often used for `ActionListeners`
- ❑ An Inner class is a class that is declared **inside** another class
 - It may be declared inside or outside a method of the class
- ❑ Why inner classes? Two reasons:
 - 1) It places the trivial listener class exactly where it is needed, without cluttering up the remainder of the project
 - 2) Their methods can access variables that are declared in surrounding blocks.
 - In this regard, inner classes declared inside methods behave similarly to nested blocks



Example Inner Class Listener

- The inner class `ClickListener` declared inside the class `ButtonFrame2` can access local variables inside the surrounding scope

Outer Block

Inner Block

```
public class ButtonFrame2 extends JFrame
{
    private JButton button;
    private JLabel label;
    ...
    class ClickListener implements ActionListener
    {
        public void actionPerformed(ActionEvent event)
        {
            label.setText("I was clicked");
        }
    }
    ...
}
```

Can easily access methods of the private instance of a label object.



ButtonFrame2.java (1)

```
1 import java.awt.event.ActionEvent;
2 import java.awt.event.ActionListener;
3 import javax.swing.JButton;
4 import javax.swing.JFrame;
5 import javax.swing.JLabel;
6 import javax.swing.JPanel;
7
8 public class ButtonFrame2 extends JFrame
9 {
10     private JButton button;
11     private JLabel label;
12
13     private static final int FRAME_WIDTH = 300;
14     private static final int FRAME_HEIGHT = 100;
15
16     public ButtonFrame2()
17     {
18         createComponents();
19         setSize(FRAME_WIDTH, FRAME_HEIGHT);
20     }
21 }
```



ButtonFrame2.java (2)

- Changes label from “Hello World!” to “I was clicked.”:

```
22  /**
23   * An action listener that changes the label text.
24  */
25  class ClickListener implements ActionListener
26  {
27      public void actionPerformed(ActionEvent event)
28      {
29          label.setText("I was clicked.");
30      }
31  }
32
33  private void createComponents()
34  {
35      button = new JButton("Click me!");
36      ActionListener listener = new ClickListener();
37      button.addActionListener(listener);
38
39      label = new JLabel("Hello, World!");
40
41      JPanel panel = new JPanel();
42      panel.add(button);
43      panel.add(label);
44      add(panel);
45  }
46 }
```



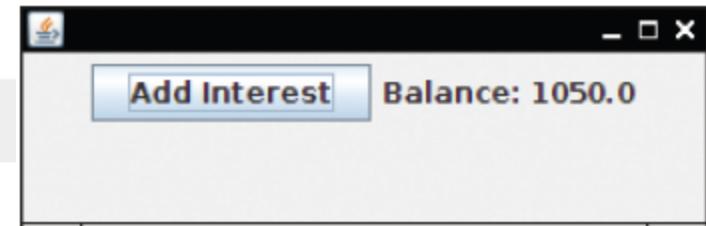
InvestmentFrame.java (1)

```
1 import java.awt.event.ActionEvent;
2 import java.awt.event.ActionListener;
3 import javax.swing.JButton;
4 import javax.swing.JFrame;
5 import javax.swing.JLabel;
6 import javax.swing.JPanel;
7
8 public class InvestmentFrame extends JFrame
9 {
10     private JButton button;
11     private JLabel resultLabel;
12     private double balance;
13
14     private static final int FRAME_WIDTH = 300;
15     private static final int FRAME_HEIGHT = 100;
16
17     private static final double INTEREST_RATE = 5;
18     private static final double INITIAL_BALANCE = 1000;
19
20     public InvestmentFrame()
21     {
22         balance = INITIAL_BALANCE;
23
24         createComponents();
25         setSize(FRAME_WIDTH, FRAME_HEIGHT);
26     }
```



InvestmentFrame.java (2)

```
28  /**
29   * Adds interest to the balance and updates the display.
30  */
31 class AddInterestListener implements ActionListener
32 {
33     public void actionPerformed(ActionEvent event)
34     {
35         double interest = balance * INTEREST_RATE / 100;
36         balance = balance + interest;
37         resultLabel.setText("Balance: " + balance);
38     }
39 }
40
41 private void createComponents()
42 {
43     button = new JButton("Add Interest");
44     ActionListener listener = new AddInterestListener();
45     button.addActionListener(listener);
46
47     resultLabel = new JLabel("Balance: " + balance);
48
49     JPanel panel = new JPanel();
50     panel.add(button);
51     panel.add(resultLabel);
52     add(panel);
53 }
54 }
```



- User clicks the button four times for output:



Common Error 10.1



□ Modifying Parameter Types in the Implementing Method

```
public interface ActionListener
{
    void actionPerformed(ActionEvent event);
}
```

- When you implement an interface, you must declare each method exactly as it is specified in the interface.
- Accidentally making small changes to the parameter types is a common error: For example:

```
class MyListener implements ActionListener
{
    public void actionPerformed()
        // Oops . . . forgot ActionEvent parameter
    {
        . . .
    }
}
```



Common Error 10.2



❑ Forgetting to Attach a Listener

- If you run your program and find that your buttons seem to be dead, double-check that you attached the button listener.
- The same holds for other user-interface components. It is a surprisingly common error to program the listener class and the event handler action without actually attaching the listener to the event source.

...

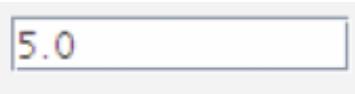
```
ActionListener listener = new ClickListener();  
button.addActionListener(listener);
```



10.3 Processing Text Input



- Dialog boxes allows for user input... but
- Popping up a separate dialog box for each input is not a natural user interface
 - Most graphical programs collect text input through text fields
 - The `JTextField` class provides a text field
 - When you construct a text field, supply the width:
 - The approximate number of characters that you expect
 - If the user exceeds this number, text will ‘scroll’ left



```
final int FIELD_WIDTH = 10;  
final JTextField rateField = new JTextField(FIELD_WIDTH);
```



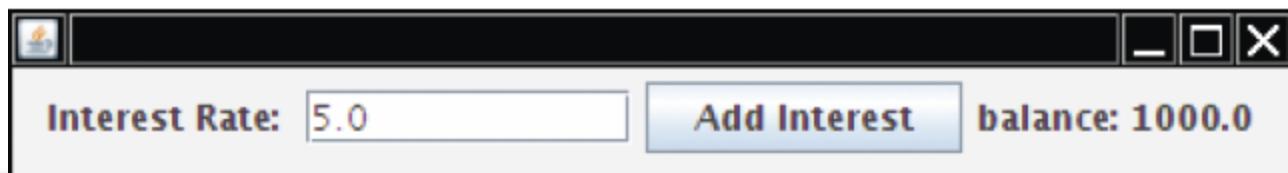
Add a Label and a Button

- A Label helps the user know what you want
 - Normally to the left of a textbox

Interest Rate: 5.0

```
JLabel rateLabel = new JLabel("Interest Rate: ");
```

- A Button with an actionPerformed method can be used to read the text from the textbox with the `getText` method
 - Note that `getText` returns a String, and must be converted to a numeric value if it will be used in calculations



```
double rate = Double.parseDouble(rateField.getText());  
double interest = account.getBalance() * rate / 100;  
account.deposit(interest);  
resultLabel.setText("balance: " + account.getBalance());
```



InvestmentFrame2.java

```
9  /**
10   * A frame that shows the growth of an investment with variable interest.
11  */
12 public class InvestmentFrame2 extends JFrame
13 {
14     private static final int FRAME_WIDTH = 450;
15     private static final int FRAME_HEIGHT = 100;
16
17     private static final double DEFAULT_RATE = 5;
18     private static final double INITIAL_BALANCE = 1000;
19
20     private JLabel rateLabel;
21     private JTextField rateField;
22     private JButton button;
23     private JLabel resultLabel;
24     private double balance;
25
26     public InvestmentFrame2()
27     {
28         balance = INITIAL_BALANCE;
29
30         resultLabel = new JLabel("Balance: " + balance);
31
32         createTextField();
33         createButton();
34         createPanel();
35     }
```

□ Use this as a framework for GUIs that do calculations

Place input components into the frame



InvestmentFrame2.java (2)

```
39 private void createTextField()
40 {
41     rateLabel = new JLabel("Interest Rate: ");
42
43     final int FIELD_WIDTH = 10;
44     rateField = new JTextField(FIELD_WIDTH);
45     rateField.setText("") + DEFAULT_RATE);
46 }
47
48 /**
49     Adds interest to the balance and updates the display.
50 */
51 class AddInterestListener implements ActionListener
52 {
53     public void actionPerformed(ActionEvent event)
54     {
55         double rate = Double.parseDouble(rateField.getText());
56         double interest = balance * rate / 100;
57         balance = balance + interest;
58         resultLabel.setText("Balance: " + balance);
59     }
60 }
61
62 private void createButton()
63 {
64     button = new JButton("Add Interest");
65
66     ActionListener listener = new AddInterestListener();
67     button.addActionListener(listener);
68 }
```

Do calculations in
ActionPerformed method

Keep the code for the
listener and the object
(Button) in the same
area



Text Areas

- Create multi-line text areas with a `JTextArea` object
 - Set the size in rows and columns

```
final int ROWS = 10;  
final int COLUMNS = 30;  
JTextArea textArea = new JTextArea(ROWS, COLUMNS);
```

- Use the `setText` method to set the text of a text field or text area

```
textArea.setText("Account Balance");
```



Text Areas

- The `append` method adds text to the end of a text area
 - Use newline characters to separate lines

```
textArea.append(account.getBalance() + "\n");
```

- Use the `setEditable` method to control user input

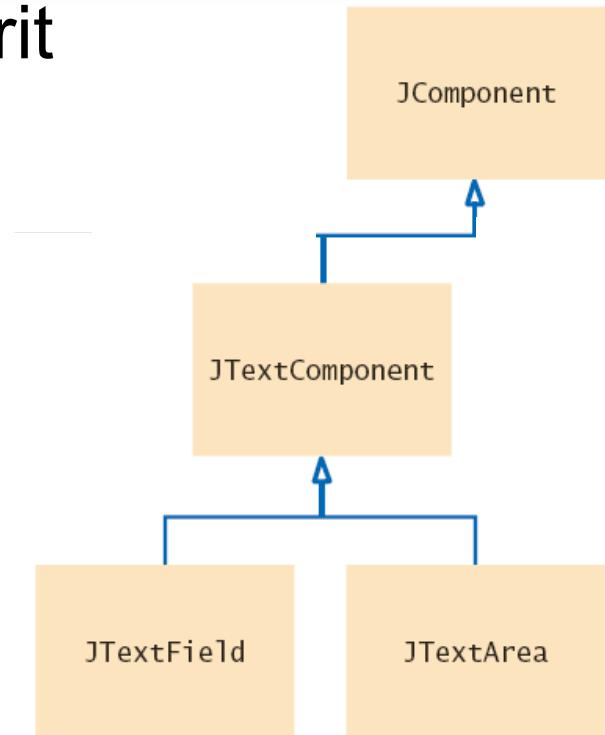
```
textArea.setEditable(false);
```



JTextField and JTextArea

- ❑ JTextField and JTextArea inherit from JTextComponent:

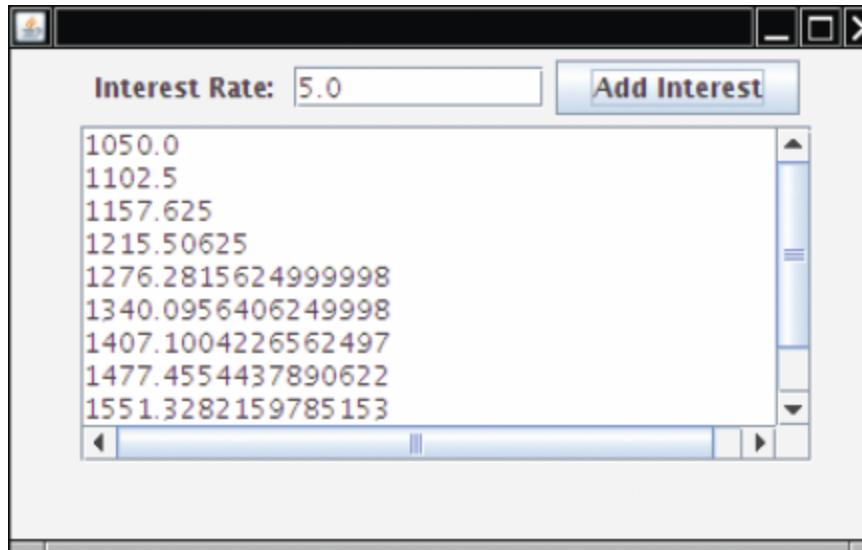
- setText
- setEditable





JTextField and JTextArea

- The append method is declared in the JTextArea class



- To add scroll bars, use JScrollPane:

```
JScrollPane scrollPane = new JScrollPane(textArea);
```



InvestmentFrame3.java (1)

```
11  /**
12   * A frame that shows the growth of an investment with variable interest,
13   * using a text area.
14  */
15 public class InvestmentFrame3 extends JFrame
16 {
17     private static final int FRAME_WIDTH = 400;
18     private static final int FRAME_HEIGHT = 250;
19
20     private static final int AREA_ROWS = 10;
21     private static final int AREA_COLUMNS = 30;
22
23     private static final double DEFAULT_RATE = 5;
24     private static final double INITIAL_BALANCE = 1000;
25
26     private JLabel rateLabel;
27     private JTextField rateField;
28     private JButton button;
29     private JTextArea resultArea;
30     private double balance;
31 }
```

Declare the components to be used



InvestmentFrame.java (2)

```
32 public InvestmentFrame3()
33 {
34     balance = INITIAL_BALANCE;
35     resultArea = new JTextArea(AREA_ROWS, AREA_COLUMNS);
36     resultArea.setText(balance + "\n");
37     resultArea.setEditable(false);
38
39     createTextField();
40     createButton();
41     createPanel();
42
43     setSize(FRAME_WIDTH, FRAME_HEIGHT);
44 }
```

Constructor calls methods to
create the components

```
46     private void createTextField()
47     {
48         rateLabel = new JL
49
50         final int FIELD_WI
51         rateField = new JT
52         rateField.setText(
53     }
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74     private void createPanel()
75     {
76         JPanel = new JPanel();
77         panel.add(rateLabel);
78         panel.add(rateField);
79         panel.add(button);
80         JScrollPane scrollPane = new JScrollPane(resultArea);
81         panel.add(scrollPane);
82         add(panel);
83     }
84 }
```



InvestmentFrame.java (3)

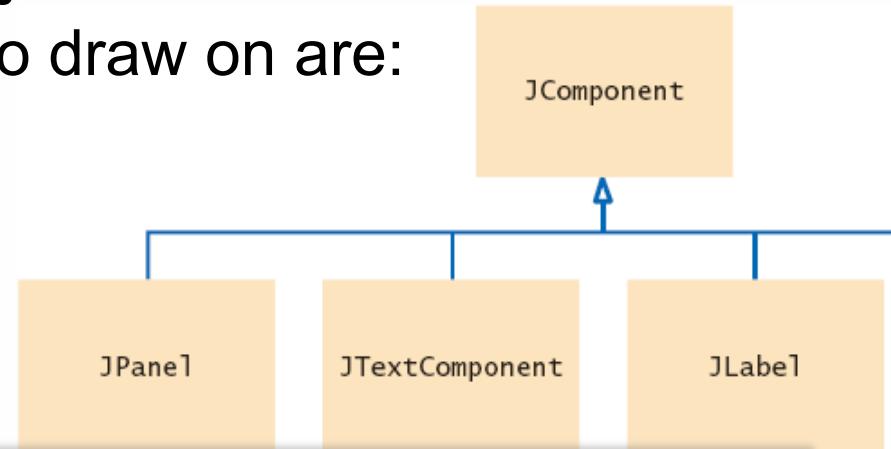
```
54  
55     class AddInterestListener implements ActionListener  
56     {  
57         public void actionPerformed(ActionEvent event)  
58         {  
59             double rate = Double.parseDouble(rateField.getText());  
60             double interest = balance * rate / 100;  
61             balance = balance + interest;  
62             resultArea.append(balance + "\n");  
63         }  
64     }  
65  
66     private void createButton()  
67     {  
68         button = new JButton("Add Interest");  
69  
70         ActionListener listener = new AddInterestListener();  
71         button.addActionListener(listener);  
72     }
```

The listener class and associated createButton method



10.4 Creating Drawings

- You cannot draw directly on a JFrame object
- Instead, construct an object and add it to the frame
 - A few examples objects to draw on are:
 - JComponent
 - JPanel
 - JTextComponent
 - JLabel



```
public class chartComponent extends JComponent
{
    public void paintComponent(Graphics g)
    {
        // Drawing instructions go here
    }
}
```

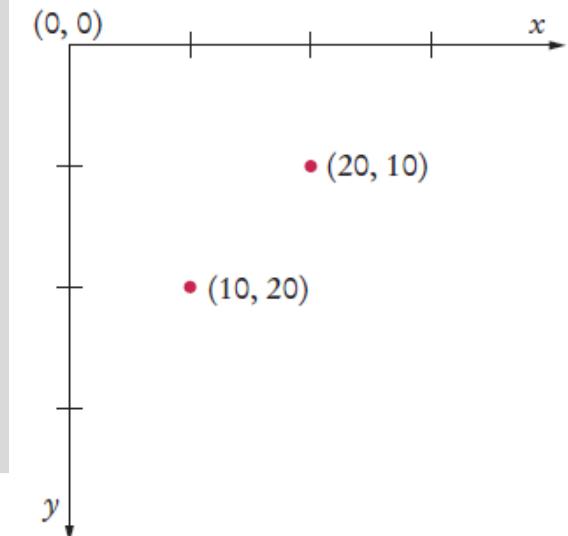
Extend the JComponent Class and override its paintComponent method



The paintComponent method

- The `paintComponent` method is called automatically when:
 - The component is shown for the first time
 - Every time the window is resized, or after being hidden

```
public class chartComponent extends JPanel  
{  
    public void paintComponent(Graphics g)  
    {  
        g.fillRect(0, 10, 200, 10);  
        g.fillRect(0, 30, 300, 10);  
        g.fillRect(0, 50, 100, 10);  
    }  
}
```





ChartComponent.java

```
1 import java.awt.Graphics;
2 import javax.swing.JComponent;
3
4 /**
5     A component that draws a bar chart.
6 */
7 public class ChartComponent extends JComponent
8 {
9     public void paintComponent(Graphics g)
10    {
11        g.fillRect(0, 10, 200, 10);
12        g.fillRect(0, 30, 300, 10);
13        g.fillRect(0, 50, 100, 10);
14    }
15 }
```

The Graphics class is part of the
java.awt package

- We now have a JComponent object that can be added to a JFrame



ChartViewer.java

```
1 import javax.swing.JComponent;
2 import javax.swing.JFrame;
3
4 public class ChartViewer
5 {
6     public static void main(String[] args)
7     {
8         JFrame frame = new JFrame();
9
10        frame.setSize(400, 200);
11        frame.setTitle("A bar chart");
12        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13
14        JComponent component = new ChartComponent();
15        frame.add(component);
16
17        frame.setVisible(true);
18    }
19}
```

Adding the component to the frame



The Graphics parameter

- The `paintComponent` method receives an object of type **Graphics**
 - The **Graphics** object stores the graphics state
 - The current color, font, etc., that are used for drawing operations
 - The **Graphics2D** class extends the **Graphics** class
 - Provides more powerful methods to draw 2D objects
 - When using Swing, the **Graphics** parameter is actually of the **Graphics2D** type, so we need to cast it to **Graphics2D** to use it

```
public class RectangleComponent extends JComponent
{
    public void paintComponent(Graphics g)
    {
        Graphics2D g2 = (Graphics2D) g;
    }
}
```

Now you are ready to draw more complex shapes!



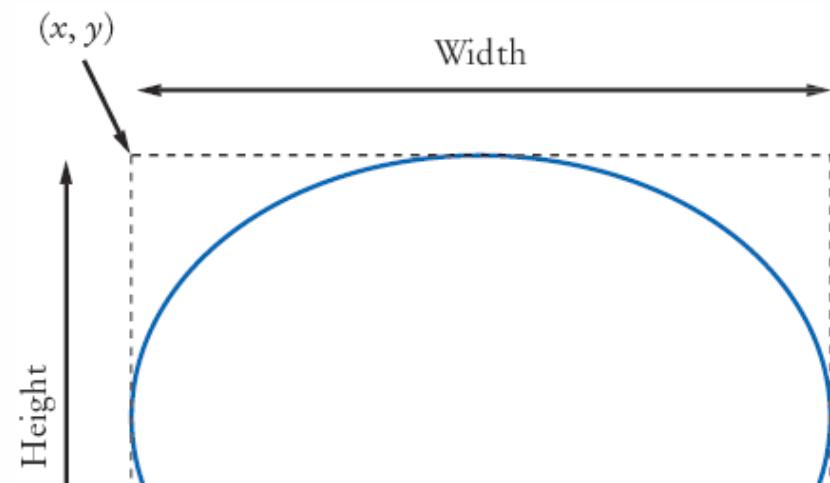
Ovals, Lines, Text, and Color

- Ellipses are drawn inside a *bounding box* in the same way that you specify a rectangle:
 - Provide the x and y coordinates of the top-left corner
 - Provide the width and height of the bounding box
 - Use the Graphics class drawOval method to create an ellipse

```
g.drawOval(x, y, width, height);
```

- Use drawLine between two points:

```
g.drawLine(x1, y1, x2, y2);
```





Drawing Text

- Use the `drawString` method of the `Graphics` class to draw a string anywhere in a window
 - Specify the String
 - Specify the Basepoint (x and y coordinates)
 - The Baseline is the y coordinate of the Basepoint

```
g2.drawString("Message", 50, 100);
```





Using Color

- ❑ All shapes and strings are drawn with a black pen and white fill by default
- ❑ To change the color, call `setColor` with an object of type `Color`
 - Java uses the RGB color model
 - You can use predefined colors, or create your own

```
g.setColor(Color.YELLOW);
g.fillOval(350, 25, 35, 20);
```

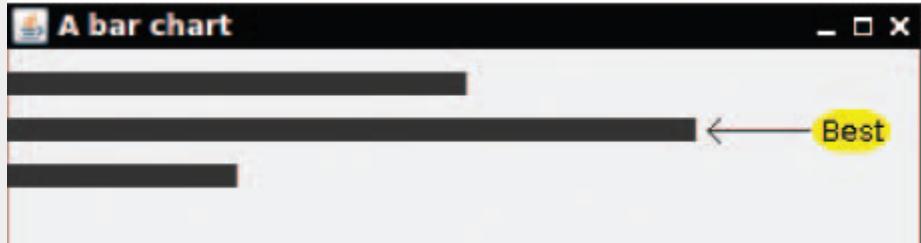
Color	RGB Value
Color.BLACK	0, 0, 0
Color.BLUE	0, 0, 255
Color.CYAN	0, 255, 255
Color.GRAY	128, 128, 128
Color.DARKGRAY	64, 64, 64
Color.LIGHTGRAY	192, 192, 192
Color.GREEN	0, 255, 0
Color.MAGENTA	255, 0, 255
Color.ORANGE	255, 200, 0
Color.PINK	255, 175, 175
Color.RED	255, 0, 0
Color.WHITE	255, 255, 255
Color.YELLOW	255, 255, 0

All shapes drawn after `setColor` will use it



ChartComponent2.java

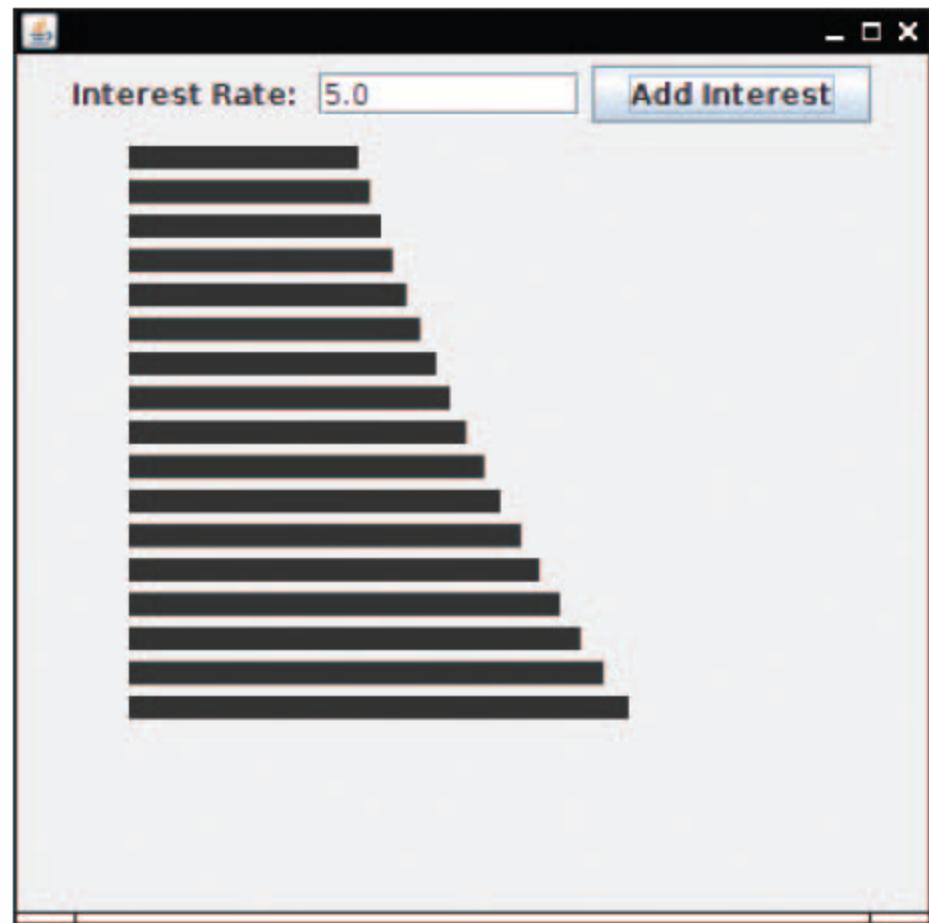
```
1 import java.awt.Color;
2 import java.awt.Graphics;
3 import javax.swing.JComponent;
4
5 /**
6     A component that draws a demo chart.
7 */
8 public class ChartComponent2 extends JComponent
9 {
10    public void paintComponent(Graphics g)
11    {
12        // Draw the bars
13        g.fillRect(0, 10, 200, 10);
14        g.fillRect(0, 30, 300, 10);
15        g.fillRect(0, 50, 100, 10);
16
17        // Draw the arrow
18        g.drawLine(350, 35, 305, 35);
19        g.drawLine(305, 35, 310, 30);
20        g.drawLine(305, 35, 310, 40);
21
22        // Draw the highlight and the text
23        g.setColor(Color.YELLOW);
24        g.fillRect(350, 25, 35, 20);
25        g.setColor(Color.BLACK);
26        g.drawString("Best", 355, 40);
27    }
28}
```





Application: Investment Growth

- ❑ Input the interest rate
- ❑ Click on the Add Interest button to add bars to the graph
- ❑ Maintains a list of values to redraw all bars each time





ChartComponent.java

```
6  /**
7   * A component that draws a chart.
8  */
9  public class ChartComponent extends JPanel
10 {
11     private ArrayList<Double> values;
12     private double maxValue;
13
14     public ChartComponent(double max)
15     {
16         values = new ArrayList<Double>();
17         maxValue = max;
18     }
19
20     public void append(double value)
21     {
22         values.add(value);
23         repaint();
24     }
25
26     public void paintComponent(Graphics g)
27     {
28         final int GAP = 5;
29         final int BAR_HEIGHT = 10;
30
31         int y = GAP;
32         for (double value : values)
33         {
34             int barWidth = (int) (getWidth() * value / maxValue);
35             g.fillRect(0, y, barWidth, BAR_HEIGHT);
36             y = y + BAR_HEIGHT + GAP;
37         }
38     }
39 }
```

Add a new value to Arraylist

Use an ArrayList to hold bar values

Paint bars in a loop



InvestmentFrame4.java (1)

```
10 /**
11  * A frame that shows the growth of an investment with variable interest,
12  * using a bar chart.
13 */
14 public class InvestmentFrame4 extends JFrame
15 {
16     public InvestmentFrame4()
17     {
18         balance = INITIAL_BALANCE;
19         chart = new ChartComponent(3 * INITIAL_BALANCE);
20         chart.setPreferredSize(new Dimension(CHART_WIDTH, CHART_HEIGHT));
21         chart.append(INITIAL_BALANCE);
22
23         createTextField();
24         createButton();
25         createPanel();
26
27         setSize(FRAME_WIDTH, FRAME_HEIGHT);
28     }
29
30
31     private void createTextField()
32     {
33         rateLabel = new JLabel("Interest Rate: ");
34
35         final int FIELD_WIDTH = 10;
36         rateField = new JTextField(FIELD_WIDTH);
37         rateField.setText("" + DEFAULT_RATE);
38     }
39
40
41
42
43
44
45
46
47
48
49
50
51
52 }
```

Instantiates and initializes
ChartComponent

Use helper methods to
create components



InvestmentFrame4.java (2)

```
54     class AddInterestListener implements ActionListener
55     {
56         public void actionPerformed(ActionEvent event)
57         {
58             double rate = Double.parseDouble(rateField.getText());
59             double interest = balance * rate / 100;
60             balance = balance + interest;
61             chart.append(balance);
62         }
63     }
64
65     private void createButton()
66     {
67         button = new JButton("Add Interest");
68
69         ActionListener listener = new AddInterestListener();
70         button.addActionListener(listener);
71     }
72
73     private void createPanel()
74     {
75         JPanel panel = new JPanel();
76         panel.add(rateLabel);
77         panel.add(rateField);
78         panel.add(button);
79         panel.add(chart);
80         add(panel);
81     }
82 }
```

Listener and
Button setup



Common Error 10.3



❑ Forgetting to Repaint

- When you change the data in a painted component, the component is not automatically painted with the new data.
- You must call the **repaint** method of the component
- The best place to call **repaint** is in the method of your component that modifies the data values:

```
void changeData(. . .)
{
    // Update data values
    repaint();
}
```



Common Error 10.4



- ❑ By default, Components have zero width and height
 - You must be careful when you add a painted component, such as a component displaying a chart, to a panel
 - The default size for a JComponent is 0 by 0 pixels, and the component will not be visible.
 - The remedy is to call the `setPreferredSize` method:

```
chart.setPreferredSize(new Dimension(CHART_WIDTH, CHART_HEIGHT));
```



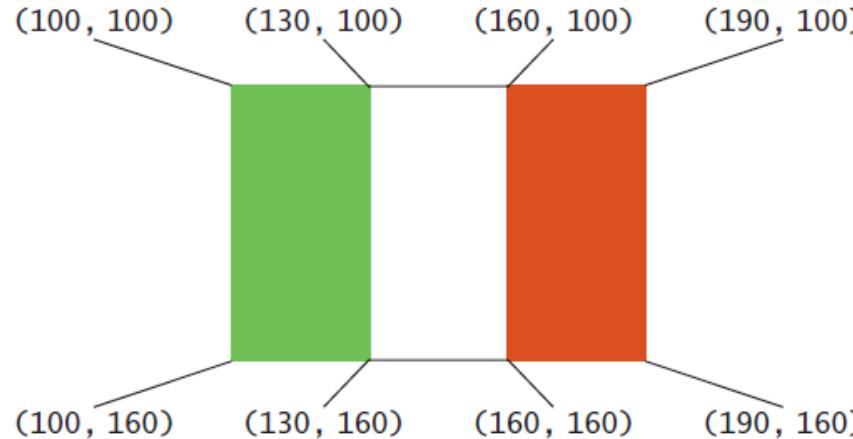
Steps to Drawing Shapes

1) Determine the shapes you need for your drawing.

- Squares and rectangles
- Circles and ellipses
- Lines and Text

2) Find the coordinates of each shape.

- For rectangles and ellipses, you need the top-left corner, width, and height of the bounding box.
- For lines, you need the x- and y-positions of the starting point and the end point.
- For text, you need the x- and y-position of the basepoint





Steps to Drawing Shapes

3) Write Java statements to draw the shapes.

```
g.setColor(Color.GREEN);
g.fillRect(100, 100, 30, 60);
g.setColor(Color.RED);
g.fillRect(160, 100, 30, 60);
10.4 Creating Drawings 499
g.setColor(Color.BLACK);
g.drawLine(130, 100, 160, 100);
g.drawLine(130, 160, 160, 160);
```

- If possible, use variables and ‘offsets’ for the locations and sizes

```
g.fillRect(xLeft, yTop, width / 3, width * 2 / 3);
. . .
g.fillRect(xLeft + 2 * width / 3, yTop, width / 3, width * 2 / 3);
. . .
g.drawLine(xLeft + width / 3, yTop, xLeft + width * 2 / 3, yTop);
```



Steps to Drawing Shapes

4) Consider using methods or classes for repetitive steps.

```
void drawItalianFlag(Graphics g, int xLeft, int yTop, int width)
{
    // Draw a flag at the given location and size
}
```

5) Place the drawing instructions in the paintComponent method.

```
public class ItalianFlagComponent extends JComponent
{
    public void paintComponent(Graphics g)
    {
        // Drawing instructions
    }
}
```

If the drawing is complex, use call methods of Step 4



Steps to Drawing Shapes

6) Write the viewer class.

Provide a viewer class, with a main method in which you construct a frame, add your component, and make your frame visible.

```
public class ItalianFlagViewer
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();
        frame.setSize(300, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JComponent component = new ItalianFlagComponent();
        frame.add(component);
        frame.setVisible(true);
    }
}
```



Summary: Frames and Components

- ❑ To show a frame, construct a `JFrame` object, set its size, and make it visible.
- ❑ Use a `JPanel` to group multiple user-interface components together.
- ❑ Declare a `JFrame` subclass for a complex frame.



Summary: Events and Handlers

- User-interface events include key presses, mouse moves, button clicks, menu selections, and so on.
- An event listener belongs to a class created by the application programmer.
 - Its methods describe the actions to be taken when an event occurs.
 - Event sources report on events. When an event occurs, the event source notifies all event listeners.
- Attach an `ActionListener` to each button so that your program can react to button clicks.
- Methods of an inner class can access variables from the surrounding class.



Summary: `TextFields` and `TextAreas`

- ❑ Use `JTextField` components to provide space for user input.
 - Place a `JLabel` next to each text field
- ❑ Use a `JTextArea` to show multiple lines of text
 - You can add scroll bars to any component with a `JScrollPane`
- ❑ You can add scroll bars to any component with a `JScrollPane`.



Summary: Simple Shapes

- In order to display a drawing, provide a class that extends the `JComponent` class.
- Place drawing instructions inside the `paintComponent` method.
 - That method is called whenever the component needs to be repainted.
- The `Graphics` class has methods to draw rectangles and other shapes.
 - Use `drawRect`, `drawOval`, and `drawLine` to draw geometric shapes.
 - The `drawString` method draws a string, starting at its basepoint.



Summary: Color and repaint

- ❑ When you set a new color in the graphics context, it is used for subsequent drawing operations.
- ❑ Call the `repaint` method whenever the state of a painted component changes.
- ❑ When placing a painted component into a panel, you need to specify its preferred size.