CHAPTER 10

GRAPHICAL USER INTERFACES

Slides by Donald W. Smith
TechNeTrain.com

Final Draft
10/30/11

Chapter Goals

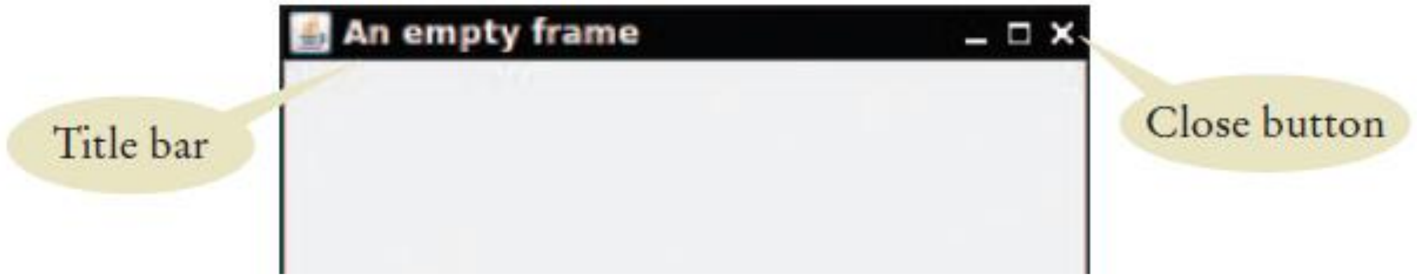
- To implement simple graphical user interfaces
- To add buttons, text fields, and other components to a frame window
- To handle events that are generated by buttons
- To write programs that display simple drawings

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

10.1 Frame Windows

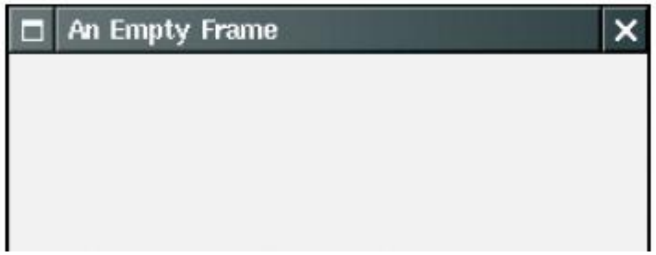
- Java provides classes to create graphical applications that can run on any major graphical user interface
  - A graphical application shows information inside a frame: a window with a title bar
- Java's `JFrame` class allows you to display a frame
  - It is part of the `javax.swing` package



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

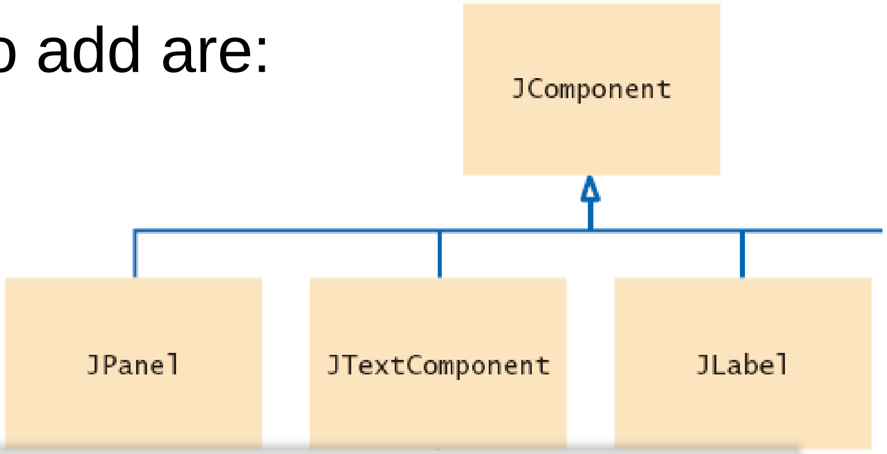
- Your JVM (Java Virtual Machine does all of the work of displaying the frame on your GUI
  - This application is portable to all supported GUIs!

```java
import javax.swing.JFrame;

/**
   This program displays an empty frame.
*/
public class EmptyFrameViewer
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();

      final int FRAME_WIDTH = 300;
      final int FRAME_HEIGHT = 400;
      frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
      frame.setTitle("An empty frame");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

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

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

```java
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

/**
   This program shows a frame that is filled with two components.
*/
public class FilledFrameViewer
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();

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

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

      final int FRAME_WIDTH = 300;
      final int FRAME_HEIGHT = 100;
      frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
      frame.setTitle("A frame with two components");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

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

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;
   private JLabel label;
   private static final int FRAME_WIDTH = 300;
   private static final int FRAME_HEIGHT = 100;

   public FilledFrame()
   {
      createComponents();
      setSize(FRAME_WIDTH, FRAME_HEIGHT);
   }
}
```

Components are instance variables

Initialize and add them in the 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

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
   An action listener that prints a message.
*/
public class ClickListener implements ActionListener
{
   public void actionPerformed(ActionEvent event)
   {
      System.out.println("I was clicked.");
   }
}
```

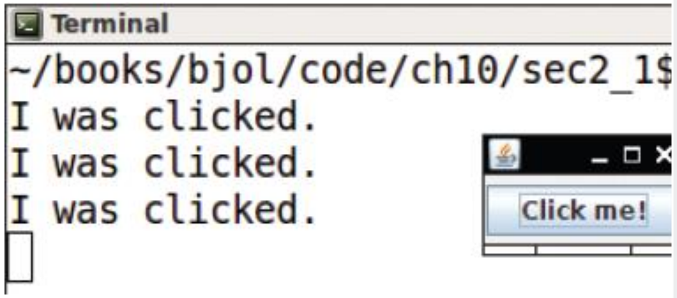
We can ignore the event parameter – it has information such as when the event occurred

Registering `ActionListener`

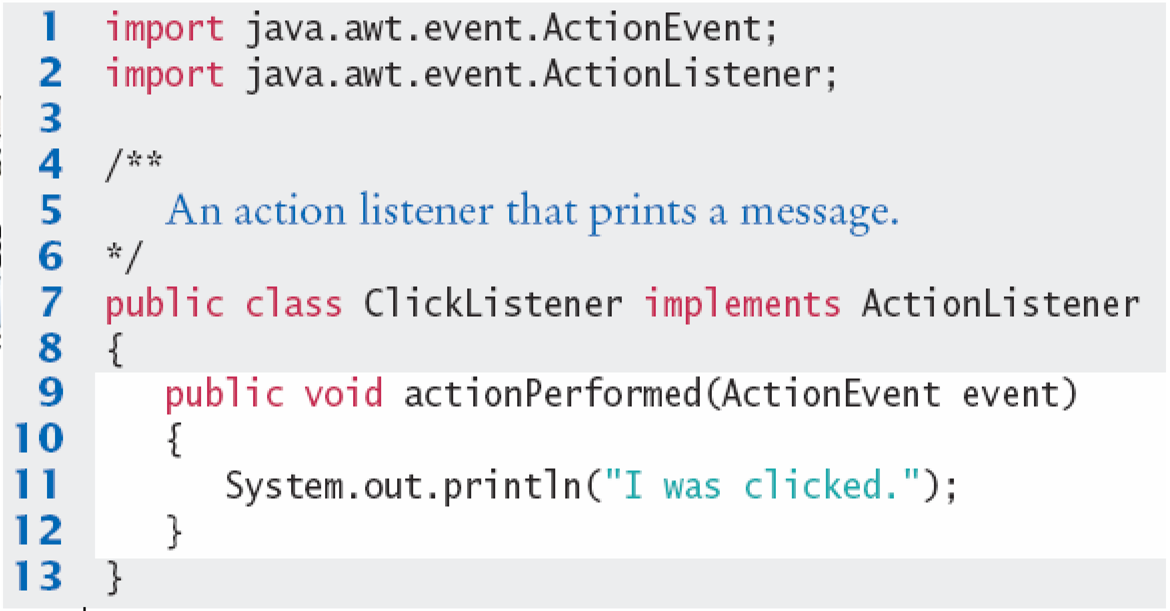
- A `ClickListener` object must be created, and then 'registered' (added) to a specific event source

```
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



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



```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
   An action listener that prints a message.
*/
public class ClickListener implements ActionListener
{
   public void actionPerformed(ActionEvent event)
   {
      System.out.println("I was clicked.");
   }
}
```

ButtonFrame1.java

```java
/**
   This frame demonstrates how to install an action listener.
*/
public class ButtonFrame1 extends JFrame
{
   private static final int FRAME_WIDTH = 100;
   private static final int FRAME_HEIGHT = 60;

   public ButtonFrame1()
   {
      createComponents();
      setSize(FRAME_WIDTH, FRAME_HEIGHT);
   }

   private void createComponents()
   {
      JButton button = new JButton("Click me!");
      JPanel panel = new JPanel();
      panel.add(button);
      add(panel);

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

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

```java
import javax.swing.JFrame;

/**
   This program demonstrates how to install an action listener.
*/
public class ButtonViewer1
{
   public static void main(String[] args)
   {
      JFrame frame = new ButtonFrame1();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
   }
}
```

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 `ActionListener`s
- 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)

```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class ButtonFrame2 extends JFrame
{
   private JButton button;
   private JLabel label;

   private static final int FRAME_WIDTH = 300;
   private static final int FRAME_HEIGHT = 100;

   public ButtonFrame2()
   {
      createComponents();
      setSize(FRAME_WIDTH, FRAME_HEIGHT);
   }

```

ButtonFrame2.java (2)

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

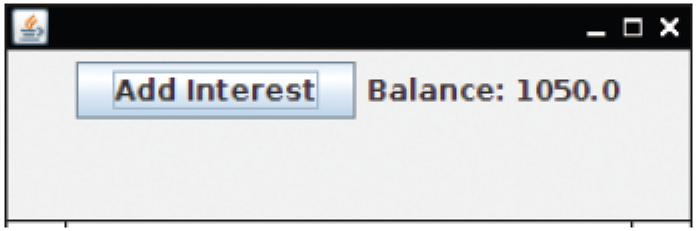
```
     /**
        An action listener that changes the label text.
     */
     class ClickListener implements ActionListener
     {
        public void actionPerformed(ActionEvent event)
        {
           label.setText("I was clicked.");
        }
     }

     private void createComponents()
     {
        button = new JButton("Click me!");
        ActionListener listener = new ClickListener();
        button.addActionListener(listener);

        label = new JLabel("Hello, World!");

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

InvestmentFrame.java (1)

```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class InvestmentFrame extends JFrame
{
   private JButton button;
   private JLabel resultLabel;
   private double balance;

   private static final int FRAME_WIDTH = 300;
   private static final int FRAME_HEIGHT = 100;

   private static final double INTEREST_RATE = 5;
   private static final double INITIAL_BALANCE = 1000;

   public InvestmentFrame()
   {
      balance = INITIAL_BALANCE;

      createComponents();
      setSize(FRAME_WIDTH, FRAME_HEIGHT);
   }
```

InvestmentFrame.java (2)

```
    /**
       Adds interest to the balance and updates the display.
    */
    class AddInterestListener implements ActionListener
    {
       public void actionPerformed(ActionEvent event)
       {
          double interest = balance * INTEREST_RATE / 100;
          balance = balance + interest;
          resultLabel.setText("Balance: " + balance);
       }
    }

    private void createComponents()
    {
       button = new JButton("Add Interest");
       ActionListener listener = new AddInterestListener();
       button.addActionListener(listener);

       resultLabel = new JLabel("Balance: " + balance);

       JPanel panel = new JPanel();
       panel.add(button);
       panel.add(resultLabel);
       add(panel);
    }
}
```



- 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
   {     . . .    }
}
```

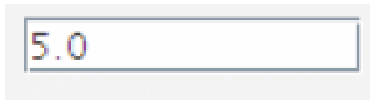
Common Error 10.2

- 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

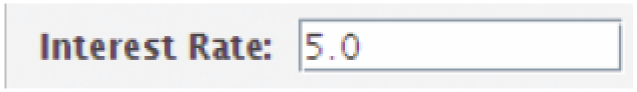
- Dialog boxes allows for user input… but
  - Popping up a separate dialog box for each input is not a natural user interface
- Most graphical programs collect text input through text fields
  - The `JTextField` class provides a text field
    - When you construct a text field, supply the width:
      - 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);
```

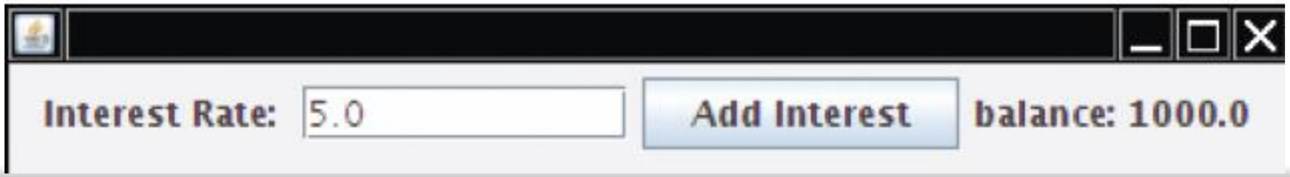
Add a Label and a Button

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



```
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

```
/**
   A frame that shows the growth of an investment with variable interest.
*/
public class InvestmentFrame2 extends JFrame
{
   private static final int FRAME_WIDTH = 450;
   private static final int FRAME_HEIGHT = 100;

   private static final double DEFAULT_RATE = 5;
   private static final double INITIAL_BALANCE = 1000;

   private JLabel rateLabel;
   private JTextField rateField;
   private JButton button;
   private JLabel resultLabel;
   private double balance;

   public InvestmentFrame2()
   {
      balance = INITIAL_BALANCE;

      resultLabel = new JLabel("Balance: " + balance);

      createTextField();
      createButton();
      createPanel();

```

- Use this as a framework for GUIs that do calculations

Place input components into the frame

InvestmentFrame2.java (2)

```
   private void createTextField()
   {
      rateLabel = new JLabel("Interest Rate: ");

      final int FIELD_WIDTH = 10;
      rateField = new JTextField(FIELD_WIDTH);
      rateField.setText("" + DEFAULT_RATE);
   }

   /**
      Adds interest to the balance and updates the display.
   */
   class AddInterestListener implements ActionListener
   {
      public void actionPerformed(ActionEvent event)
      {
         double rate = Double.parseDouble(rateField.getText());
         double interest = balance * rate / 100;
         balance = balance + interest;
         resultLabel.setText("Balance: " + balance);
      }
   }

   private void createButton()
   {
      button = new JButton("Add Interest");

      ActionListener listener = new AddInterestListener();
      button.addActionListener(listener);
   }
```

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

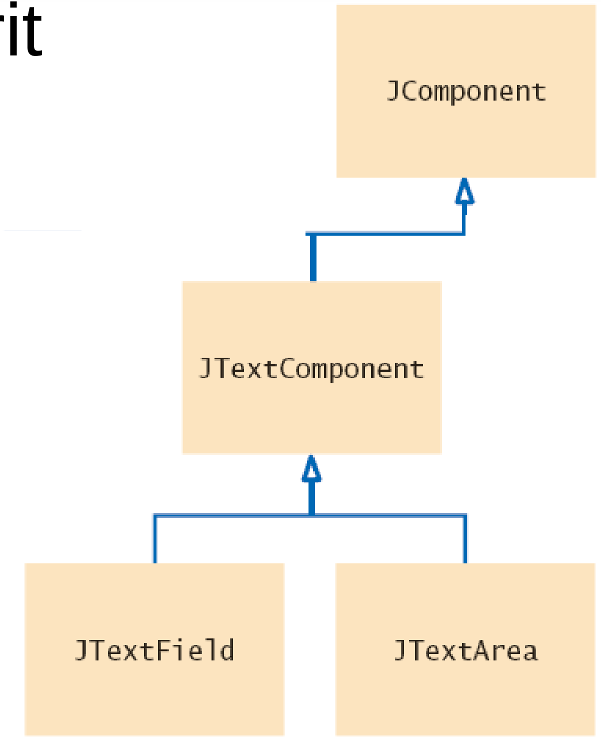
```
textArea.append(account.getBalance() + "\n");
```

- Use the `setEditable` method to control user input

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

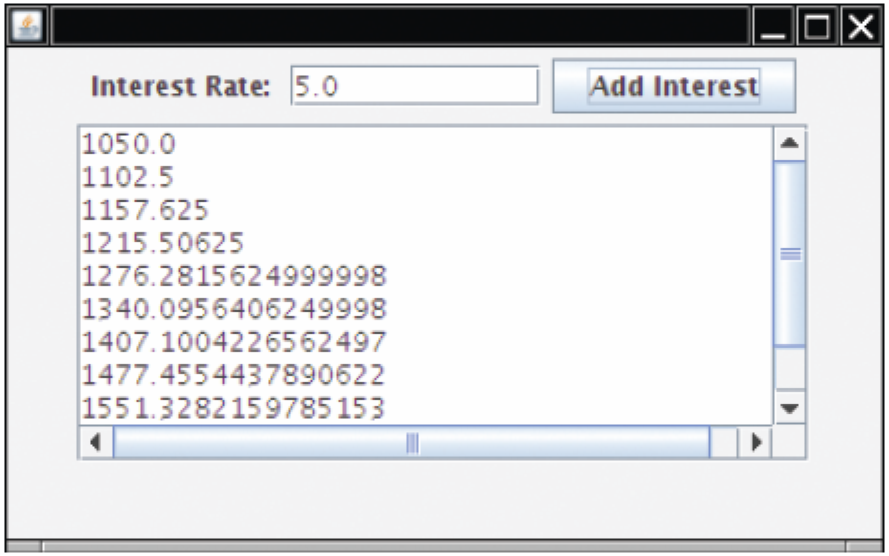
JTextField and JTextArea

- JTextField and JTextArea inherit from JTextComponent:
  - setText
  - setEditable



JTextField and JTextArea

- The append method is declared in the `JTextArea` class



```
1050.0
1102.5
1157.625
1215.50625
1276.2815624999998
1340.0956406249998
1407.1004226562497
1477.4554437890622
1551.3282159785153
```

- To add scroll bars, use `JScrollPane`:

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

InvestmentFrame3.java (1)

```
/**
   A frame that shows the growth of an investment with variable interest,
   using a text area.
*/
public class InvestmentFrame3 extends JFrame
{
   private static final int FRAME_WIDTH = 400;
   private static final int FRAME_HEIGHT = 250;

   private static final int AREA_ROWS = 10;
   private static final int AREA_COLUMNS = 30;

   private static final double DEFAULT_RATE = 5;
   private static final double INITIAL_BALANCE = 1000;

   private JLabel rateLabel;
   private JTextField rateField;
   private JButton button;
   private JTextArea resultArea;
   private double balance;

```

Declare the components to be used

InvestmentFrame.java (2)

```
    public InvestmentFrame3()
    {
       balance = INITIAL_BALANCE;
       resultArea = new JTextArea(AREA_ROWS, AREA_COLUMNS);
       resultArea.setText(balance + "\n");
       resultArea.setEditable(false);

       createTextField();
       createButton();
       createPanel();

       setSize(FRAME_WIDTH, FRAME_HEIGHT);
    }

    private void createTextField()
    {
       rateLabel = new JL

       final int FIELD_WI
       rateField = new JT
       rateField.setText(
    }
```

Constructor calls methods to create the components

```
    private void createPanel()
    {
       JPanel = new JPanel();
       panel.add(rateLabel);
       panel.add(rateField);
       panel.add(button);
       JScrollPane scrollPane = new JScrollPane(resultArea);
       panel.add(scrollPane);
       add(panel);
    }
 }
```

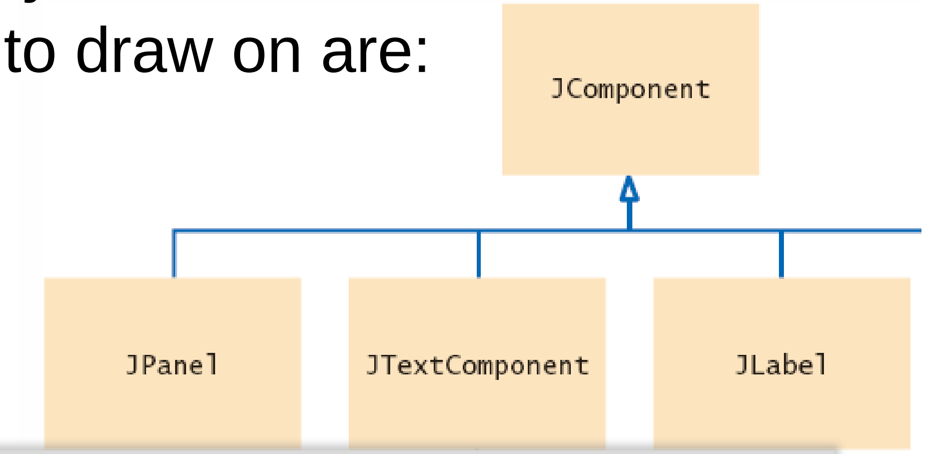
InvestmentFrame.java (3)

```

   class AddInterestListener implements ActionListener
   {
      public void actionPerformed(ActionEvent event)
      {
         double rate = Double.parseDouble(rateField.getText());
         double interest = balance * rate / 100;
         balance = balance + interest;
         resultArea.append(balance + "\n");
      }
   }

   private void createButton()
   {
      button = new JButton("Add Interest");

      ActionListener listener = new AddInterestListener();
      button.addActionListener(listener);
   }
```

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`



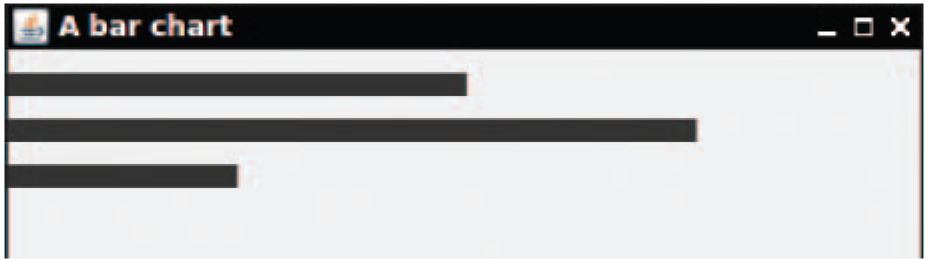
```
public class chartComponent extends JComponent
{
  public void paintComponent(Graphics g)
  {
    // Drawing instructions go here
  }
}
```

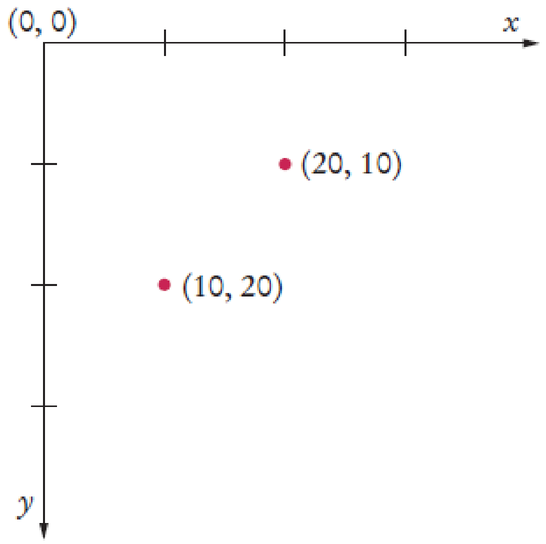
Extend the `JComponent` Class and override its `paintComponent` method

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 JComponent
{
   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

```
import java.awt.Graphics;
import javax.swing.JComponent;

/**
   A component that draws a bar chart.
*/
public class ChartComponent extends JComponent
{
   public void paintComponent(Graphics g)
   {
      g.fillRect(0, 10, 200, 10);
      g.fillRect(0, 30, 300, 10);
      g.fillRect(0, 50, 100, 10);
   }
}
```

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

```
import javax.swing.JComponent;
import javax.swing.JFrame;

public class ChartViewer
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();

      frame.setSize(400, 200);
      frame.setTitle("A bar chart");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      JComponent component = new ChartComponent();
      frame.add(component);

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

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

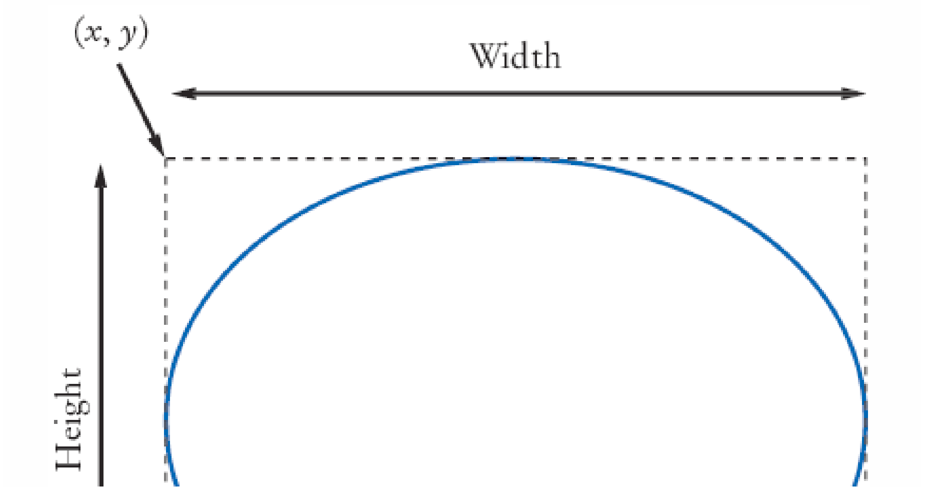
```
public class RectangleComponent extends JComponent
{
  public void paintComponent(Graphics g)
  {
    Graphics2D g2 = (Graphics2D) g;
  }
}
```

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);
```



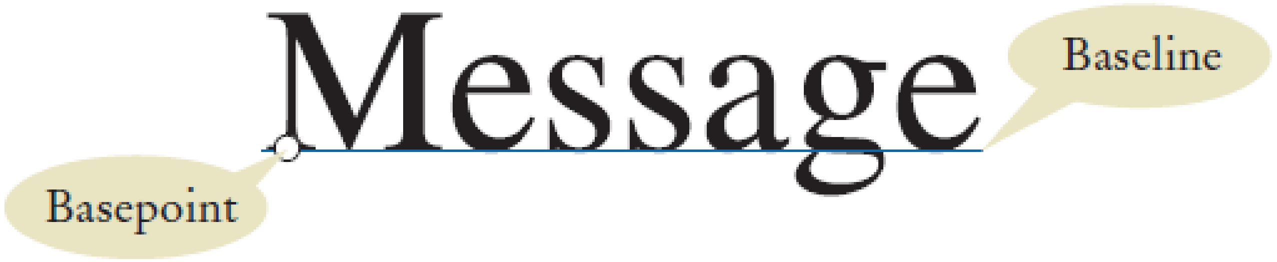
  - Use drawLine between two points:

```
g.drawLine(x1, y1, x1, 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

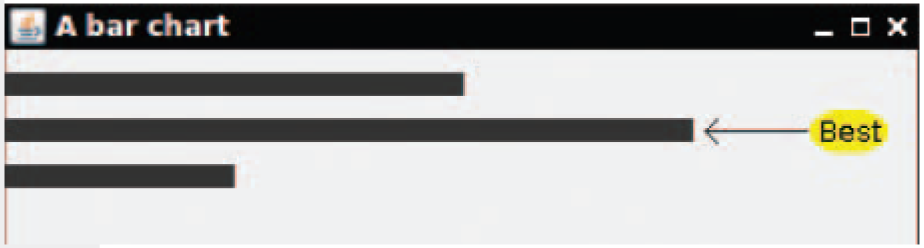
```
g.setColor(Color.YELLOW);
g.fillOval(350, 25, 35, 20);
```

All shapes drawn after `setColor` will use it

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

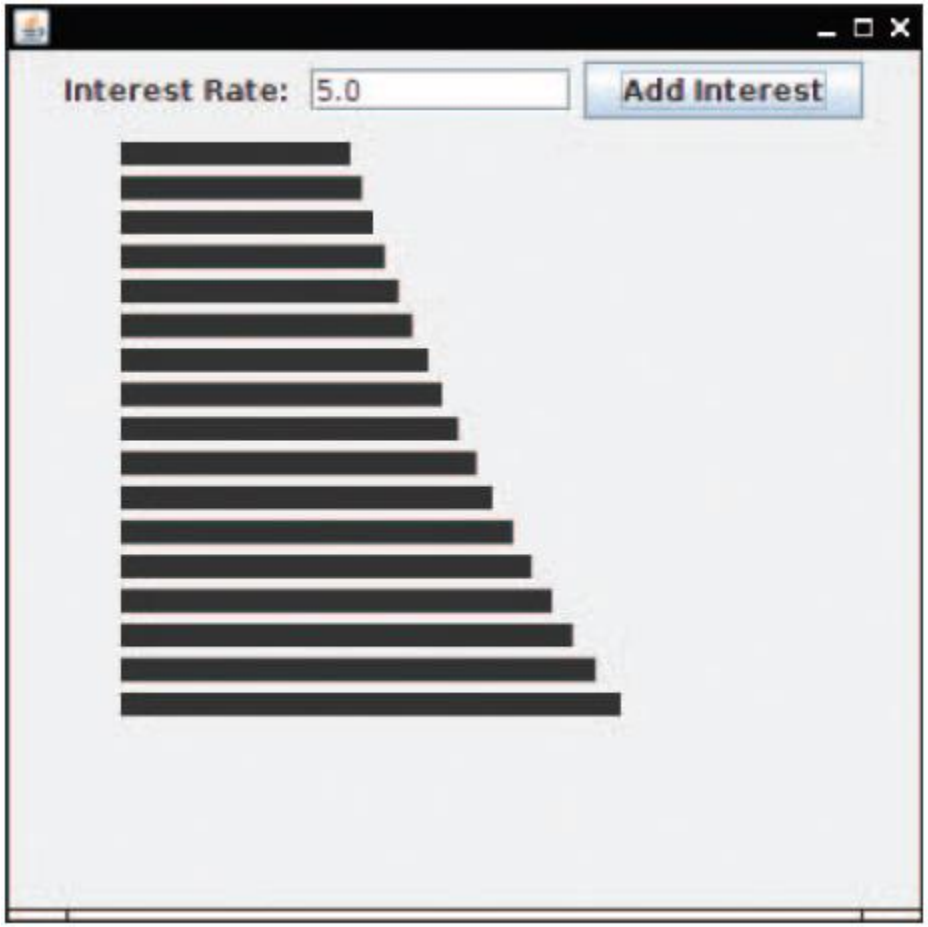
ChartComponent2.java

```java
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JComponent;

/**
   A component that draws a demo chart.
*/
public class ChartComponent2 extends JComponent
{
   public void paintComponent(Graphics g)
   {
      // Draw the bars
      g.fillRect(0, 10, 200, 10);
      g.fillRect(0, 30, 300, 10);
      g.fillRect(0, 50, 100, 10);

      // Draw the arrow
      g.drawLine(350, 35, 305, 35);
      g.drawLine(305, 35, 310, 30);
      g.drawLine(305, 35, 310, 40);

      // Draw the highlight and the text
      g.setColor(Color.YELLOW);
      g.fillOval(350, 25, 35, 20);
      g.setColor(Color.BLACK);
      g.drawString("Best", 355, 40);
   }
}
```



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

```java
/**
   A component that draws a chart.
*/
public class ChartComponent extends JComponent
{
   private ArrayList<Double> values;
   private double maxValue;

   public ChartComponent(double max)
   {
      values = new ArrayList<Double>();
      maxValue = max;
   }

   public void append(double
   {
      values.add(value);
      repaint();
   }
```

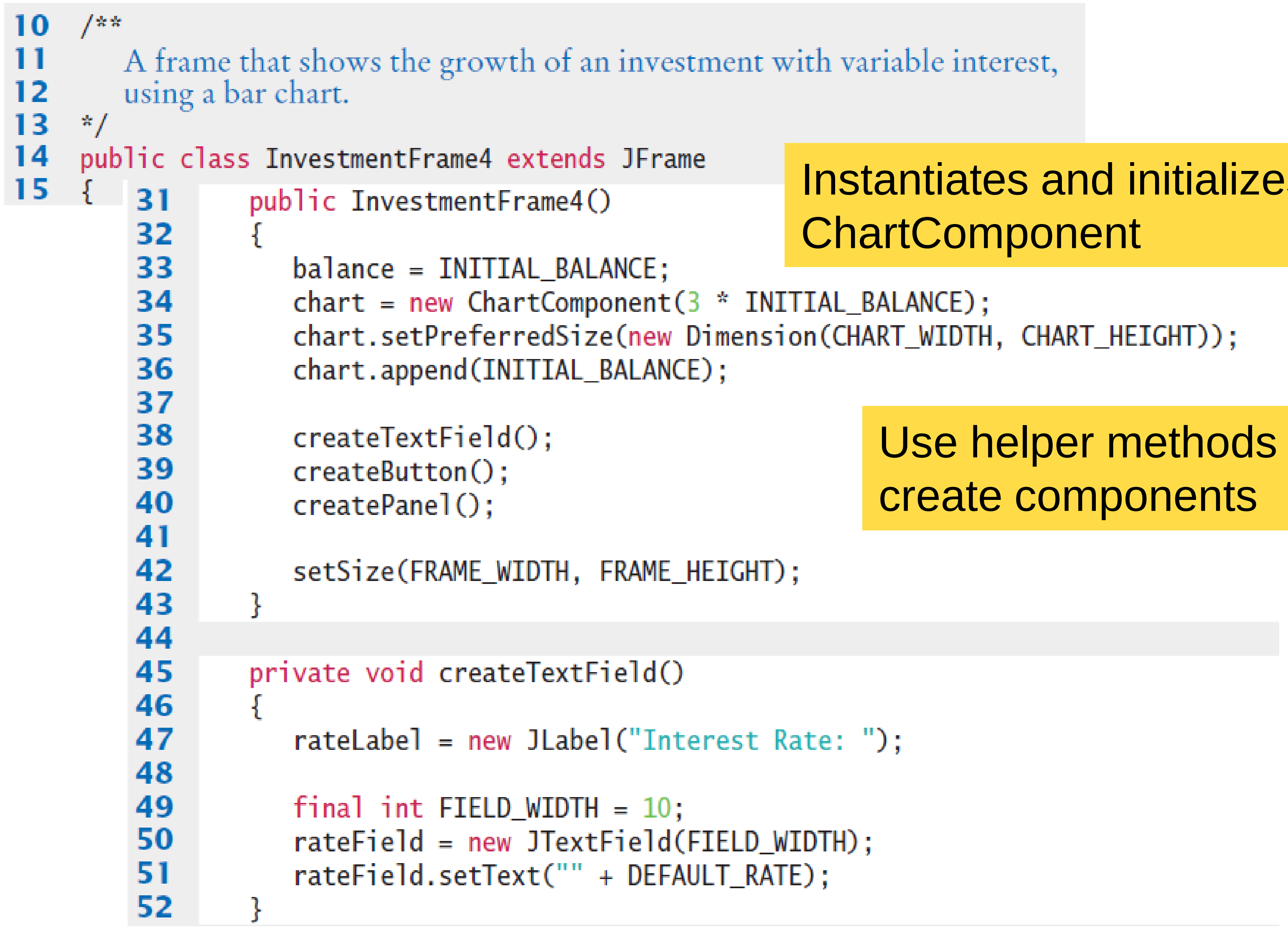
Use an ArrayList to hold bar values

Add a new value to Arraylist

```java
   public void paintComponent(Graphics g)
   {
      final int GAP = 5;
      final int BAR_HEIGHT = 10;

      int y = GAP;
      for (double value : values)
      {
         int barWidth = (int) (getWidth() * value / maxValue);
         g.fillRect(0, y, barWidth, BAR_HEIGHT);
         y = y + BAR_HEIGHT + GAP;
      }
   }
}
```

Paint bars in a loop

InvestmentFrame4.java (1)



```
/**
   A frame that shows the growth of an investment with variable interest,
   using a bar chart.
*/
public class InvestmentFrame4 extends JFrame
{
   public InvestmentFrame4()
   {
      balance = INITIAL_BALANCE;
      chart = new ChartComponent(3 * INITIAL_BALANCE);
      chart.setPreferredSize(new Dimension(CHART_WIDTH, CHART_HEIGHT));
      chart.append(INITIAL_BALANCE);

      createTextField();
      createButton();
      createPanel();

      setSize(FRAME_WIDTH, FRAME_HEIGHT);
   }

   private void createTextField()
   {
      rateLabel = new JLabel("Interest Rate: ");

      final int FIELD_WIDTH = 10;
      rateField = new JTextField(FIELD_WIDTH);
      rateField.setText("" + DEFAULT_RATE);
   }
```

Instantiates and initializes ChartComponent

Use helper methods to create components

InvestmentFrame4.java (2)

```java
   class AddInterestListener implements ActionListener
   {
      public void actionPerformed(ActionEvent event)
      {
         double rate = Double.parseDouble(rateField.getText());
         double interest = balance * rate / 100;
         balance = balance + interest;
         chart.append(balance);
      }
   }
```

```java
   private void createButton()
   {
      button = new JButton("Add Interest");

      ActionListener listener = new AddInterestListener();
      button.addActionListener(listener);
   }

   private void createPanel()
   {
      JPanel panel = new JPanel();
      panel.add(rateLabel);
      panel.add(rateField);
      panel.add(button);
      panel.add(chart);
      add(panel);
   }
}
```

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

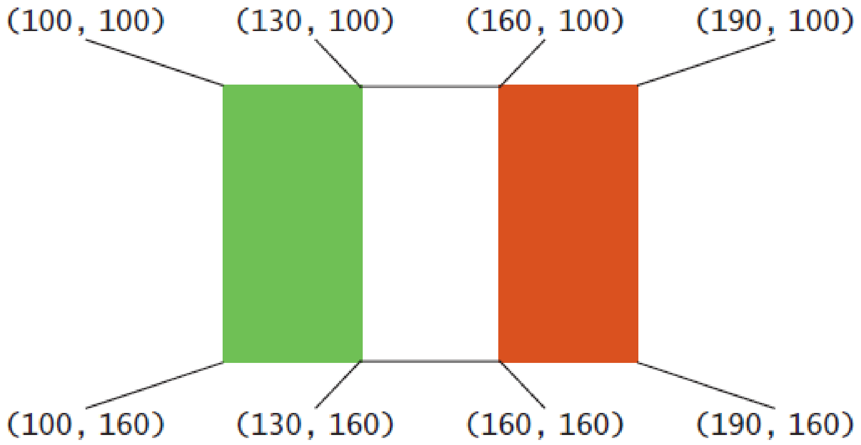
- By default, Components have zero width and height
  - You must be careful when you add a painted component, such as a component displaying a chart, to a panel
  - The default size for a `JComponent` is 0 by 0 pixels, and the component will not be visible.
  - The remedy is to call the `setPreferredSize`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