Advanced Computer Programming

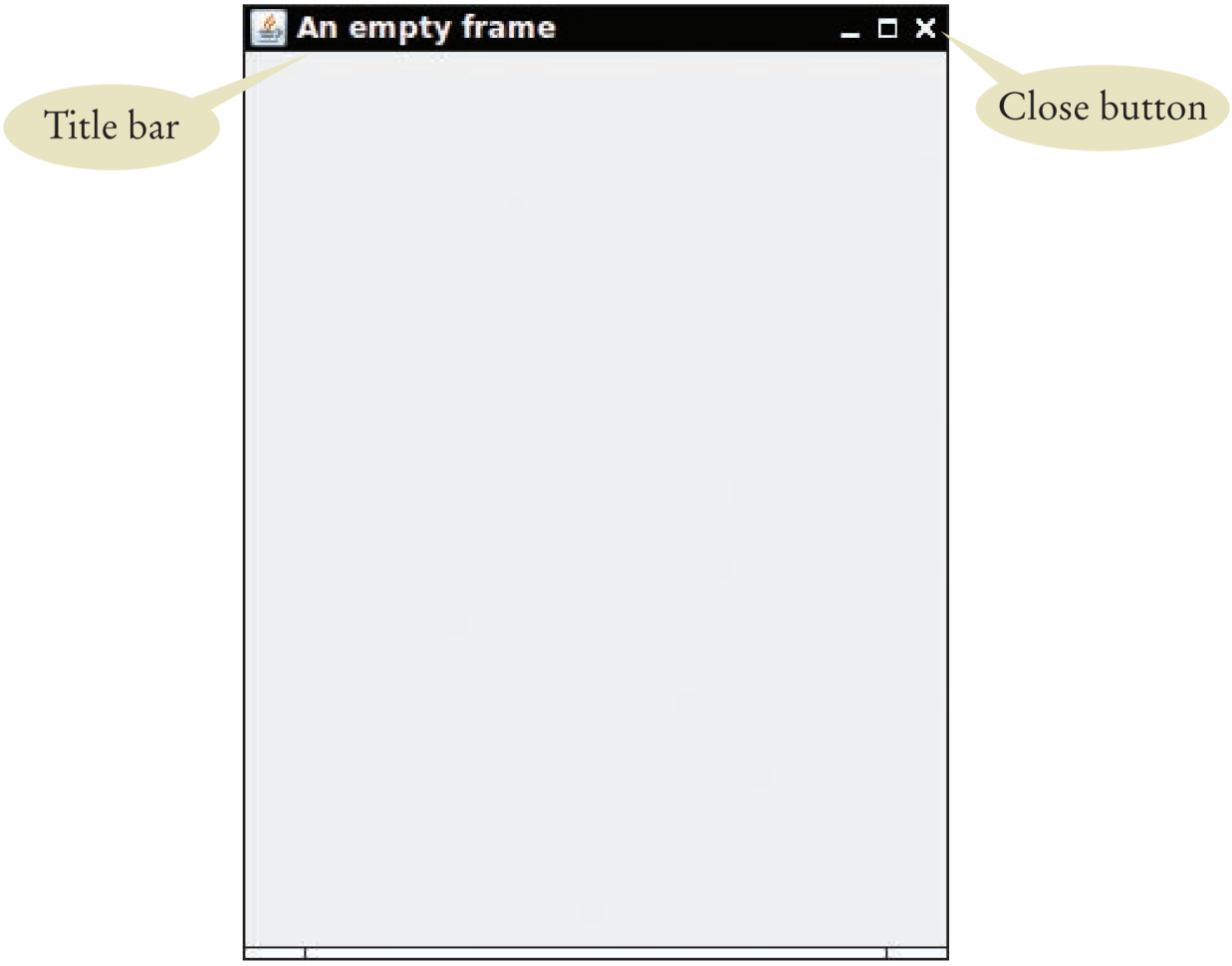
[Lecture 12]

Saeed Reza Kheradpisheh

kheradpisheh@ut.ac.ir

Department of Computer Science
Shahid Beheshti University
Spring 1397-98

Frame Windows



Frame Windows

To show a frame, carry out the following steps:

1. Construct an object of the `JFrame` class:

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

2. Set the size of the frame:

```
final int FRAME_WIDTH = 300;
final int FRAME_HEIGHT = 400;
frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
```

3. If you'd like, set the title of the frame:

```
frame.setTitle("An empty frame");
```

   If you omit this step, the title bar is simply left blank.

4. Set the "default close operation":

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

   When the user closes the frame, the program automatically exits. Don't omit this step. If you do, the program keeps running even after the frame is closed.

5. Make the frame visible:

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

```
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);
   }
}
```

Adding Components to a Frame

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:

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

You first construct the components, providing the text that should appear on them:

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

Adding Components to a Frame

```
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);
   }
}
```

Adding Components to a Frame

Using Inherience to Costumize Frames

```
public class FilledFrame extends JFrame
{
   // Use instance variables for components
   private JButton button;
   private JLabel label;

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

   public FilledFrame()
   {
      // Now we can use a helper method
      createComponents();

      // It is a good idea to set the size in the frame constructor
      setSize(FRAME_WIDTH, FRAME_HEIGHT);
   }

   private void createComponents()
   {
      button = new JButton("Click me!");
      label = new JLabel("Hello, World!");
      JPanel panel = new JPanel();
      panel.add(button);
      panel.add(label);
      add(panel);
   }
}
```

Using Inherience to Costumize Frames

```
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);
   }
}
```

Events and Event Handling

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

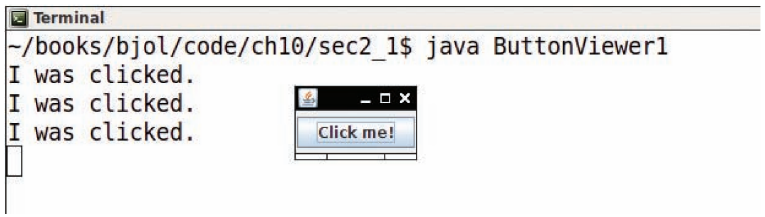
```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
/**
   An action listener that prints a message.
*/
public class ClickListener implements ActionListener
{
   public void actionPerformed(ActionEvent event)
   {
      System.out.println("I was clicked.");
   }
}
```

Events and Event Handling

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

/**
   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);
   }
}
```

Events and Event Handling

```
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);
   }
}
```



Changing the Label's Text

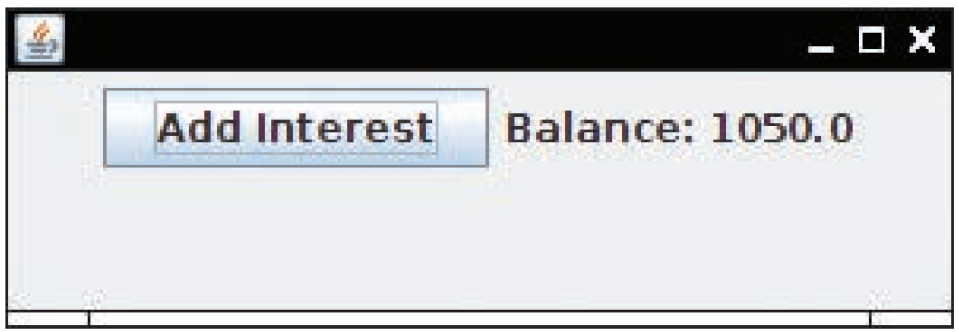
```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

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);
   }
```

Changing the Label's Text

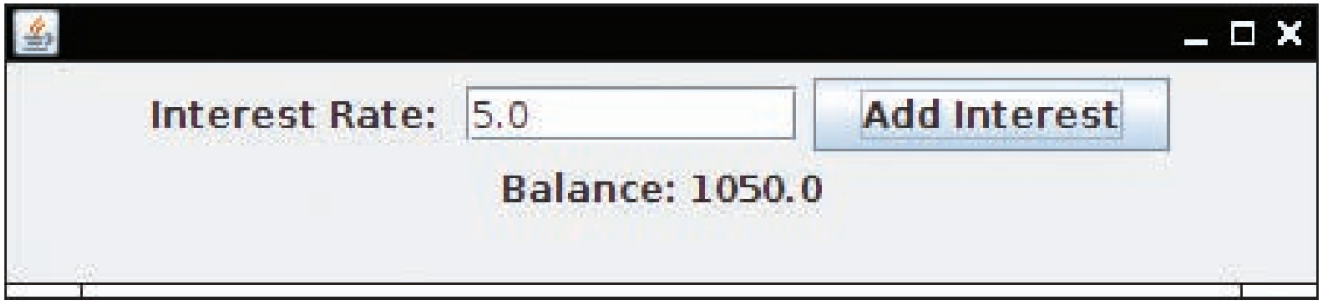
```
   /**
      An action listener that changes the label text.
   */
   class ClickListener implements ActionListener
   {
      public void actionPerformed(ActionEvent event)
      {
         label.setText("I was clicked.");
      }
   }

   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);
   }
}
```

Practice your GUI

- Make a GUI as below.
- Each time the button is clicked, an interest rate is added to the total balance.
- Start from the initial balance of 1000$.
- The label should show the total balance.



Processing text input



Processing text input

```
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;
import javax.swing.JTextField;

/**
   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();
```

Processing text input

```
      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);
   }

   /**
      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);
   }

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

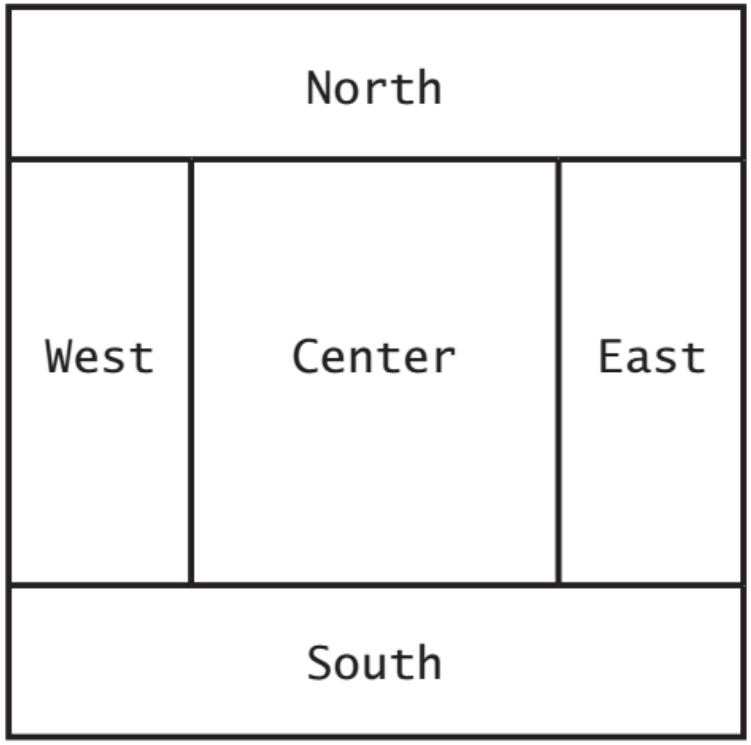
layout Management

- In Java, you build up user interfaces by adding components into containers such as panels.
- Each container has its own layout manager, by default, a **JPanel** uses a **flow layout**.
- A flow layout simply arranges its components from *left to right* and starts a new row when there is no more room in the current row.
- Another commonly used layout manager is the **border layout**.
- The border layout is the default layout manager for a frame.

```
panel.setLayout(new BorderLayout());
```

Border Layout

```
panel.setLayout(new BorderLayout());
panel.add(component, BorderLayout.NORTH);
```



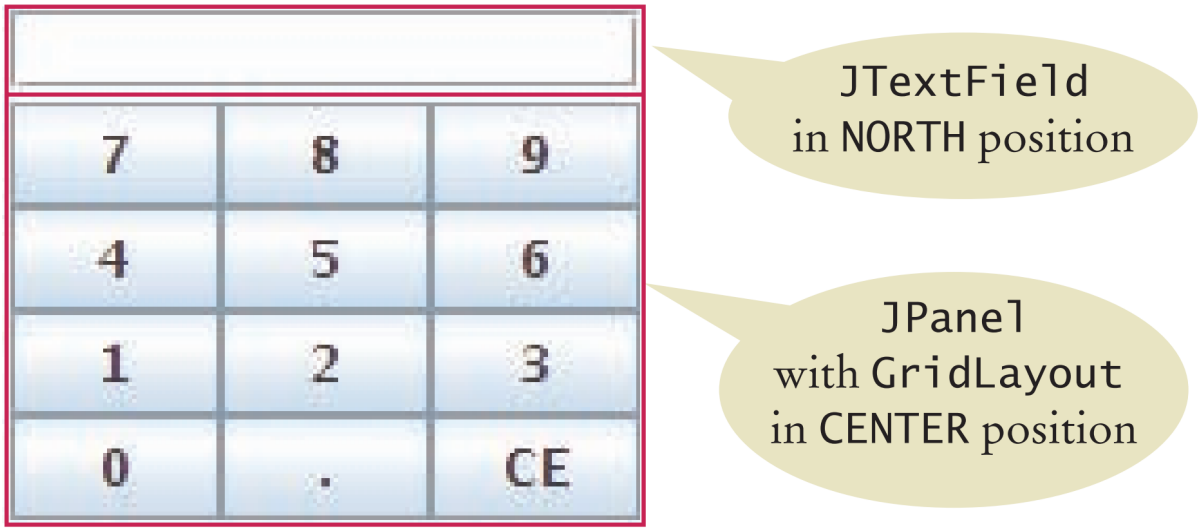
Grid Layout

- The manager arranges components in a grid with a fixed number of rows and columns.

```
JPanel buttonPanel = new JPanel();
buttonPanel.setLayout(new GridLayout(4, 3));
buttonPanel.add(button7);
buttonPanel.add(button8);
buttonPanel.add(button9);
buttonPanel.add(button4);
. . .
```

Nested Pannels

```
JPanel keypadPanel = new JPanel();
keypadPanel.setLayout(new BorderLayout());
buttonPanel = new JPanel();
buttonPanel.setLayout(new GridLayout(4, 3));
buttonPanel.add(button7);
buttonPanel.add(button8);
// . . .
keypadPanel.add(buttonPanel, BorderLayout.CENTER);
JTextField display = new JTextField();
keypadPanel.add(display, BorderLayout.NORTH);
```



Radio Buttons

If the choices are mutually exclusive, use a set of **radio buttons**.

```
JRadioButton smallButton = new JRadioButton("Small");
JRadioButton mediumButton = new JRadioButton("Medium");
JRadioButton largeButton = new JRadioButton("Large");

ButtonGroup group = new ButtonGroup();
group.add(smallButton);
group.add(mediumButton);
group.add(largeButton);
```

The `isSelected` method is called to find out whether a button is currently selected or not. For example,

```
if (largeButton.isSelected()) { size = LARGE_SIZE; }
```

Check Box

A **check box** is a user-interface component with two states: checked and unchecked.

You construct a check box by providing the name in the constructor:

```
JCheckBox italicCheckBox = new JCheckBox("Italic");
```

As with radio buttons, you use the `isSelected` method to find out whether a check box is currently checked or not.

Combo Box

If you have a large number of choices, you dont want to make a set of radio buttons. Instead, you can use a **combo box**.

```
JComboBox facenameCombo = new JComboBox();
facenameCombo.addItem("Serif");
facenameCombo.addItem("SansSerif");
. . .
```

You get the item that the user has selected by calling the `getSelectedItem` method.

```
String selectedString = (String) facenameCombo.getSelectedItem();
```

Example: Font Viewer