جَامعة المَنارة
MANARA UNIVERSITY

كلية الهندسة المعلوماتية

برمجة 3
Java Programming

ا. د. علي عمران سليمان

محاضرات الأسبوع السابع والثامن

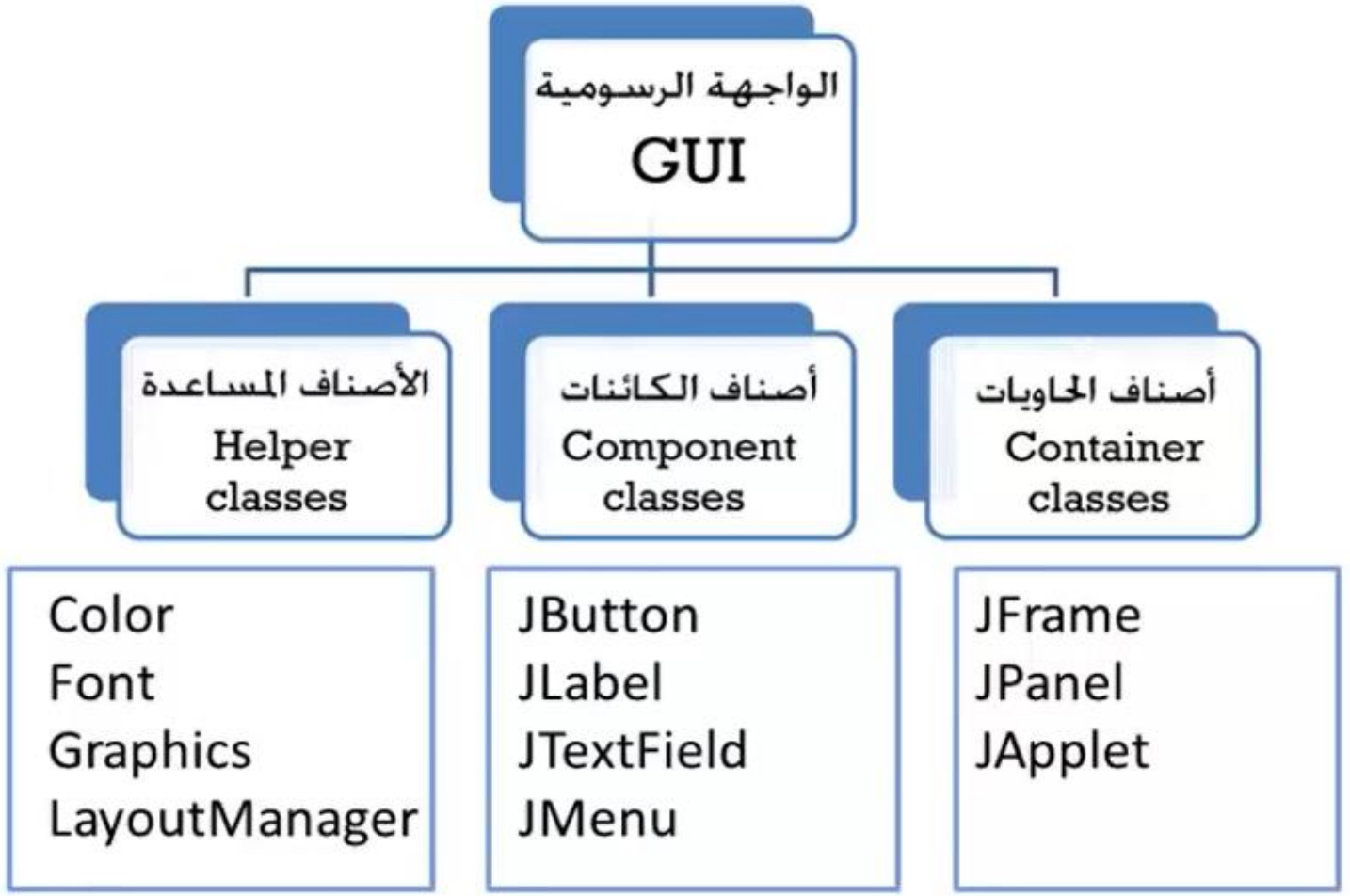
الفصل الصيفي 2023-2024

Outline

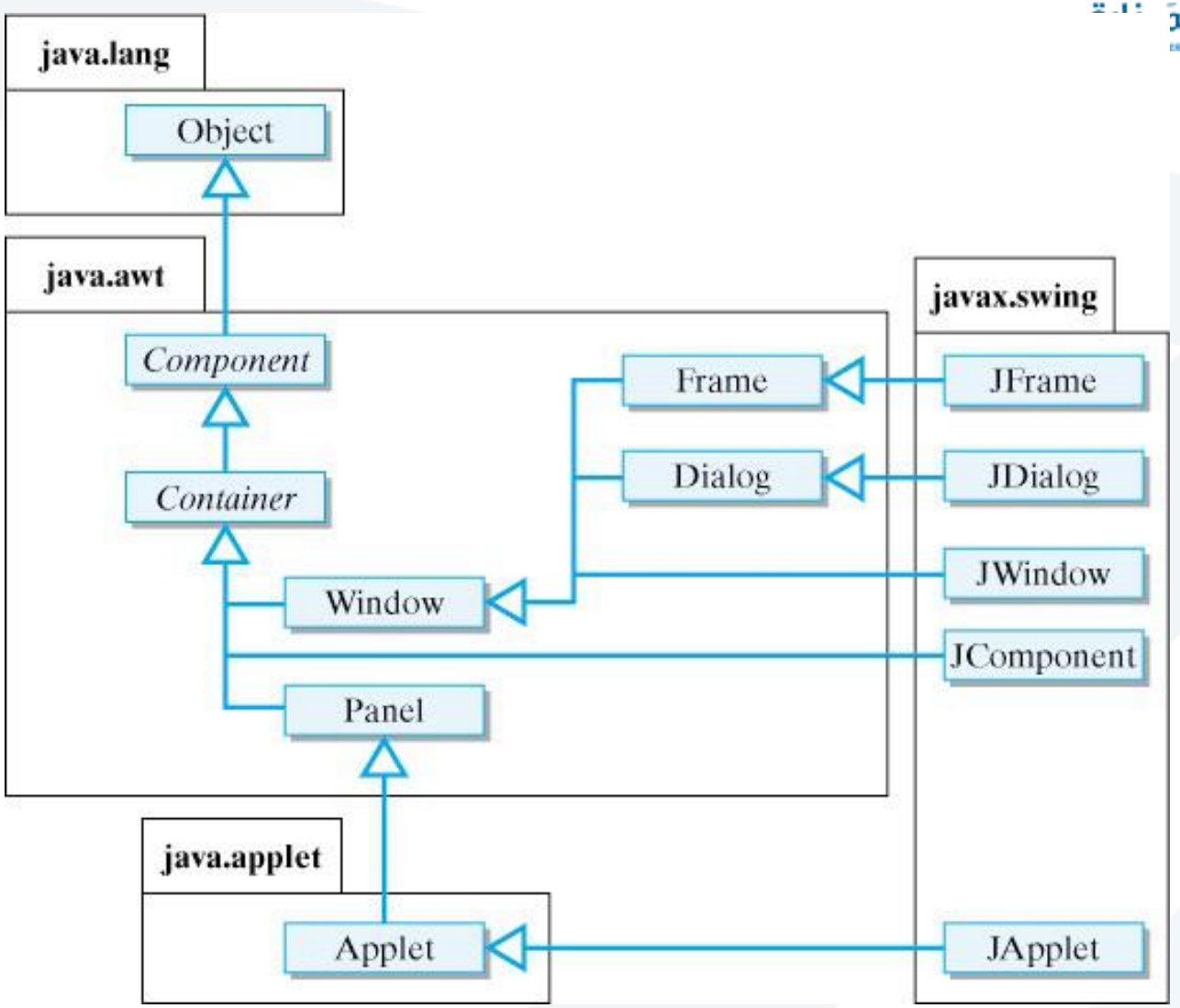
References - Deitel & Deitel, Java How to Program, Pearson; 10th Ed(2015)

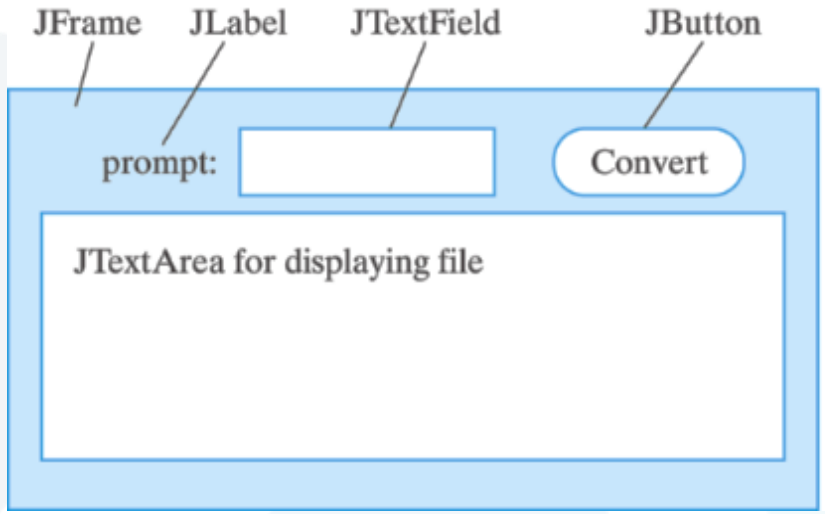
- د.علي سليمان، بنى معطيات بلغة JAVA، جامعة تشرين 2013-2014

GUI Classes


الواجهة الرسومية
GUI
الأصناف المساعدة
Helper classes
أصناف الكائنات
Component classes
أصناف الحاويات
Container classes
Color
Font
Graphics
LayoutManager
JButton
JLabel
JTextField
JMenu
JFrame
JPanel
JApplet

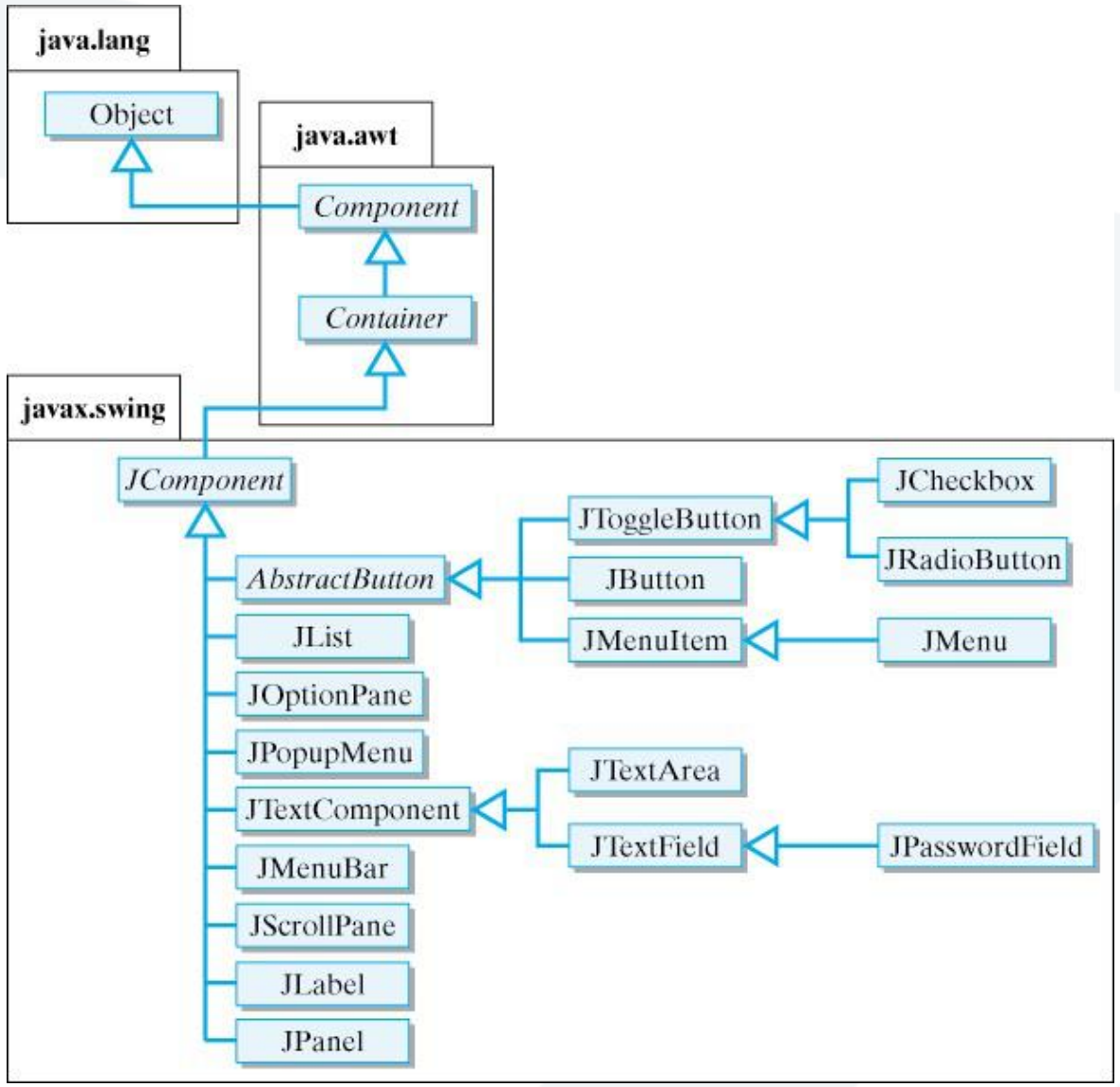
تذكرة



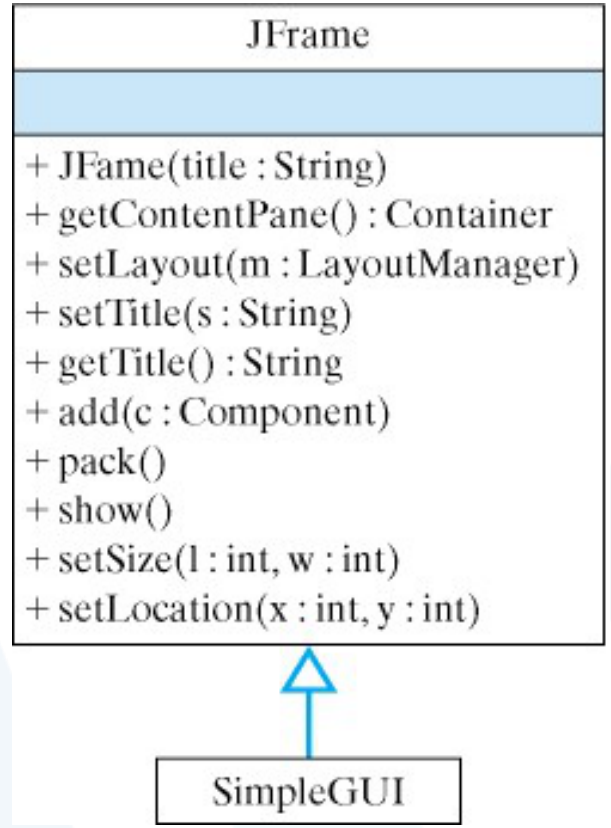


هو تسلسل هرمي للفئات يوضح العلاقات بين بعض فئات Swing وAWT ذات المستوى الأعلى. على سبيل المثال، فئة javax.swing.JFrame ، التي تمثل نافذة ذات مستوى أعلى، هي فئة فرعية من java.awt.Frame ، و javax.swing.Jpanel هي فئة فرعية من java.awt.Panel. يمكننا أن نرى من هذا الشكل أن JFrameisaFrame و FrameisaWindow و WindowisaContainer. يتم إنشاء علاقات الفئة الفرعية/الفئة العليا هذه في تعريفات الفئات الخاصة بها باستخدام كلمة المفتاح extends على النحو التالي:

Swing components



تذكرة



يوضح الباني كيفية استخدام بعض الأساليب الموروثة من JFrame. ونذكر بعض الأساليب التي يرثها SimpleGUI من JFrame. نستخدم أساليب ()setSize و ()setLocation لتعيين حجم SimpleGUI وموقعه. نستخدم أسلوب () setTitle لتعيين عنوانه. ونستخدم أسلوب () setVisible لجعله يظهر على وحدة التحكم.

تذكرة

كل برنامج رسومي يستخدم نافذة إطار frame window أو أكثر ولكل نافذة اطار شريط عنوان titel bar وحدود border لكي نظهر الإطار نستخدم الصنف JFream من الحزمة javax.swing ويجب:

1- إنشاء كائن من JFream وفق حيث First هي عنوان الإطار ;("First")JFrame appli = new JFrame ويمكن أن تترك فارغة ونستخدم appli.setTitel("First");

2- تحديد مقاس الإطار من الطريقة setSize . appli.setSize(300, 300);

3- إضافة الرسمة أو ماتم تجميعة ونرغب بعرضه إلى الإطار appli.add(panel); في حال وجودها.

4- جعل الإطار مرئي تستخدم الطريقة show لجعل مدير عرض النافذة window manager يعرضها افتراضيا هي false.

appli.setVisible(true);

5- عند تنفيذ البرنامج يتم إظهار الإطار وينتهي تنفيذ main ولكن يظل البرنامج يعمل والإطار ظاهر على الشاشة ويمكن تحريكة وتغيير حجمه و ... ، وعند إغلاق نافذة الإطار بالضغط على أيقونة الاغلاق من شريط العنونه يظل البرنامج يعمل ولايحدث شيئ سوى إختفاء الإطار،ويمكن استخدام معالجة الحدث النقر على أيقونة الغلق من أجل انهاء البرنامج إضافة على إغلاق النافذة أو إنهاء البرنامج بإستخدام System.exit(0) والتي يجب أن تكون بنهاية main ولكن تخلق مشكلة جديدة وهي ظهور النافذة للحظة وجيزة وينتهي فوراً والرغبة هي انهاء البرنامج عند يضغط المستخدم على أيقونة الغلق في شريط العنوان وهنا نجد اسهل طريقة استخدام المنهج : appli.setDefaultCloseOperation(JFrame.*EXIT_ON_CLOSE*); ولها عدة حالات مثل *DO_NOTHING_ON_CLOSE* وهنا لن يعمل شيئ عند النقر على *X*.

- **class Graphics من الحزمة** (java.awt)، **والتي توفر طرقًا مختلفة لرسم النص والأشكال على الشاشة.**
- **الصنف** JPanel **من الحزمة** (javax.swing)، **والتي توفر مساحة يمكن الرسم عليها.**

public class DrawPanel extends JPanel

extends **الكلمة الأساسية للإشارة إلى أن الصنف** DrawPanel **هو نوع محسن من** JPanel **وارث له.**

- **الكلمة الأساسية** extends **تمثل ما يسمى بعلاقة الوراثة التي يبدأ فيها صنفنا الجديد** DrawPanel **بالأعضاء الحاليين (البيانات والأساليب) من فئة** Jpanel .
- **كل لوحة** JPanel **بما في ذلك** DrawPanel، **لديها طريقة** paintComponent.
- **ينادي النظام تلقائيًا في كل مرة يحتاج فيها إلى عرض** DrawPanel. **المنهج** paintComponent() **ويجب التصريح عنها** public void paintComponent(Graphics g)، **خلاف ذلك،لن يسمح النظام بمناداتها والعبارة الأولى فيها عندما تكتبها (تحملها تحميلاً زائداً) هي** super.paintComponent(g);.
- **يتم استدعاء هذه الطريقة عندما يتم عرض** JPanel **لأول مرة على الشاشة، وعندما يتم تغطيتها** **hidden** **ثم الكشف عنها بواسطة نافذة أخرى على الشاشة، وعندما يتم تغيير حجم النافذة التي تظهر فيها.**

9.12 GUI &Graphics

Demonstrates the use of labels

- Labels هي طريقة مناسبة لتحديد مكونات واجهة المستخدم الرسومية على الشاشة وإبقاء المستخدم على اطلاع بالحالة الحالية للبرنامج.
- يمكن لـ `JLabel` من الحزمة `javax.swing` عرض نص أو صورة أو كليهما.
- يوضح المثال في الشكل **9.13** العديد من ميزات `JLabel`، بما في ذلك تسمية النص العادي وتسمية الصورة والتسمية مع كل من النص والصورة.

JLabel with text and with images

LabelDemo 1

```
package ch12GUI;
//Fig 9.13: LabelDemo.java
//Demonstrates the use of labels.
import java.awt.BorderLayout;// contains constants that specify where we can place GUI components
import javax.swing.ImageIcon;  // represents an image that can be displayed on a Jlabel.
import javax.swing.JLabel;     // to create an object can display text, an image or both.
import javax.swing.JFrame;     // represents the window that will contain all the labels.
public class LabelDemo
{public static void main( String[] args )
{  // Create a label with plain text
   JLabel northLabel = new JLabel( "North" );
      // create an icon from an image so we can put it on a JLabel
      // ImageIcon can load images in GIF, JPEG and PNG image formats.
   ImageIcon labelIcon = new ImageIcon( "GUItip.gif" );
      // create a label with an Icon instead of text
   JLabel centerLabel = new JLabel( labelIcon );
      // create another label with an Icon
   JLabel southLabel = new JLabel( labelIcon );
```

الصنف //LabelDemo

JLabel with text and with images

LabelDemo 2

```
    // set the label to display text (as well as an icon)
    southLabel.setText( "South" );
    // create a frame to hold the labels
    JFrame application = new JFrame("LabelDemo");
    application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

    // add the labels to the frame; the second argument specifies
    // where on the frame to add the label
    application.add( northLabel, BorderLayout.NORTH );
    application.add( centerLabel, BorderLayout.CENTER );
    application.add( southLabel, BorderLayout.SOUTH );

    application.setSize( 300, 300 ); // set the size of the frame
    application.setVisible( true ); // show the frame
} // end main
} // end class LabelDemo
```

الصنف LabelDemo

Demonstrates the use of labels

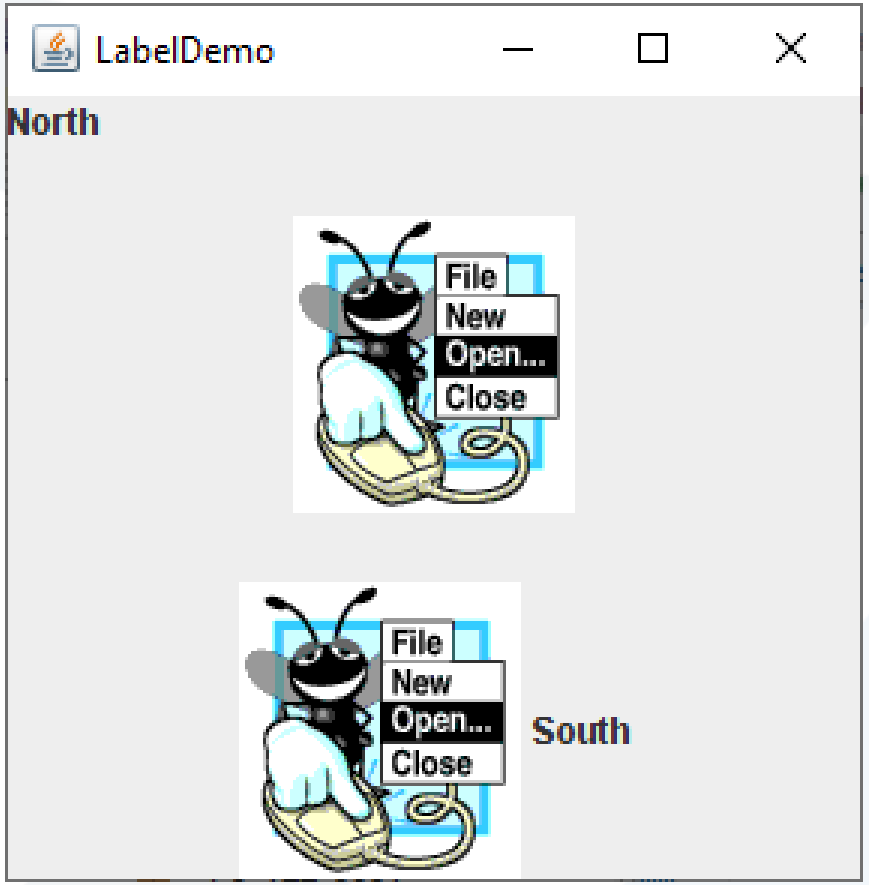


Fig. 9.13 | JLabel with text and with images

الصنف TestDraw.java

9.12 GUI &Graphics

JLabel with text and with images.

- ✓ يتلقى باني الصنف ImageIcon سلسلة تحدد المسار إلى الصورة.
- ✓ في حال تلقى اسم الملف فقط، تفترض Java أنه موجود في نفس الدليل مثل الصنف .LabelDemo.
- ✓ يمكن للصنف ImageIcon تحميل الصور بتنسيقات صورGIFو JPEGو PNG.
- ✓ تستدعي الطريقة setText لإضافة تسمية أو لتغيير النص الذي تعرضه التسمية. يمكن استدعاء الأسلوب setText على أي JLabel لتغيير نصه.
- ✓ يعرض هذا JLabel كلاً من الرمز والنص.
- ✓ من خلال استدعاء نسخة محملة بشكل زائد من طريقة إضافة تأخذ بارامترين. الأول هي المكون الذي نريد إرفاقه ، والثاني هو المنطقة التي يجب وضعها فيه.
- ✓ يحتوي كل إطار JFrameعلى تخطيط مرتبط يساعد JFrame في وضع مكونات واجهة المستخدم الرسومية المرفقة به.
- ✓ يُعرف التخطيط الافتراضي لـ JFrame باسم BorderLayout وله خمس مناطق

Five regions - North (top), South (bottom), East (right side), West (left side) and Center.

showInputDialog
showMessageDialog

- // display result in a JOptionPane message dialog

```
JOptionPane.showMessageDialog(null, "The sum is " + sum, "Sum of Two Integers",
JOptionPane.PLAIN_MESSAGE);
```

- قد تتضمن طريقة العرض الساكنه showMessageDialog من الصنف JOptionPane أربع متغيرات.
- الأول يحدد مكان ظهور مربع الحوار إذا غاب أو كان null ستكون في الوسط.
- الثاني الرسالة التي يجب عرضها - في هذه الحالة ، نتيجة تسلسل السلسلة "المجموع هو" وقيمة المجموع.
- الثالث السلسلة التي يجب أن تظهر في شريط العنوان أعلى مربع الحوار وهنا **"Sum of Two Integers** .
- الرابع الايقونه التي ستظهر على يسار مربع الحوار وإن كانت PLAIN_MESSAGE لن تظهر ايقونه.
- توجد أربع ايقونات هي QUESTION_MESSAGE، INFORMATION_MESSAGE، WARNING_MESSAGE ، ERROR_MESSAGE والحالة PLAIN_MESSAGE بدون ايقونه.

showMessageDialog

JOptionPane static constants for message dialogs

Message dialog type	Icon	Description
ERROR_MESSAGE		Indicates an error.
INFORMATION_MESSAGE		Indicates an informational message.
WARNING_MESSAGE		Warns of a potential problem.
QUESTION_MESSAGE		Poses a question. This dialog normally requires a response, such as clicking a Yes or a No button.
PLAIN_MESSAGE	no icon	A dialog that contains a message, but no icon

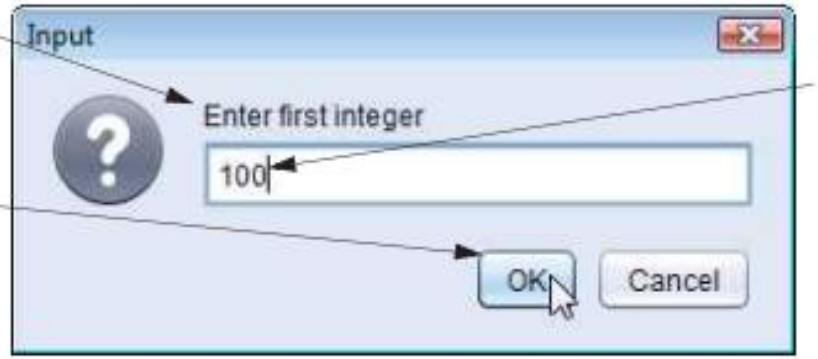
Fig. 12.3 | JOptionPane static constants for message dialogs.

program that uses JOptionPane for input and output

(a) Input dialog displayed by lines 10–11

Prompt to the user

When the user clicks **OK**, `showInputDialog` returns to the program the `100` typed by the user as a `String`; the program must convert the `String` to an `int`



Text field in which the user types a value

(b) Input dialog displayed by lines 12–13

(c) Message dialog displayed by lines 22–23—When the user clicks **OK**, the message dialog is dismissed (removed from the screen)

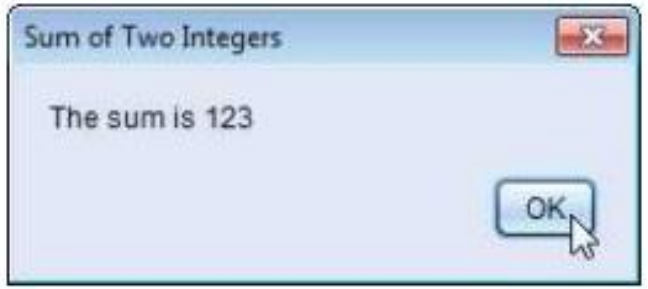


Fig. 12.2 | Addition program that uses `JOptionPane` for input and output. (Part 2 of 2.)

Some basic Swing GUI components

Component	Description
JLabel	Displays uneditable text and/or icons.
JTextField	Typically receives input from the user.
JButton	Triggers an event when clicked with the mouse.
JCheckBox	Specifies an option that can be selected or not selected.
JComboBox	A drop-down list of items from which the user can make a selection.
JList	A list of items from which the user can make a selection by clicking on any one of them. Multiple elements can be selected.
JPanel	An area in which components can be placed and organized.

Fig. 12.4 | Some basic Swing GUI components.

12.5 Displaying Text and Images Using Labels

- JFrameهو صنف فرعي غير مباشرة من الصنف java.awt.Window الذي يوفر السمات والسلوكيات الأساسية للنافذة مثل شريط العنوان في الأعلى، وأزرار لتصغير النافذة وتكبيرها وإغلاقها.
- تتكون GUI النموذجية من العديد من المكونات. غالبًا ما يقدم مصممو GUI نصًا يوضح الغرض من كل منها.
- نص التسمية يتم إنشاؤه باستخدام JLabel - صنف فرعية من JComponent.
- نادرًا ما تغير التطبيقات محتويات التسميات بعد إنشائها.
- لكل عنصر GUI العديد من الميزات أكثر مما يمكننا تغطيته في الأمثلة الخاصة بنا.
- لمعرفة التفاصيل الكاملة، قم بزيارة صفحته في التوثيق عبر الإنترنت.
- docs.oracle.com/javase/7/docs/api/javax/swing/JLabel.html.

12.5 Displaying Text and Images Using Labels

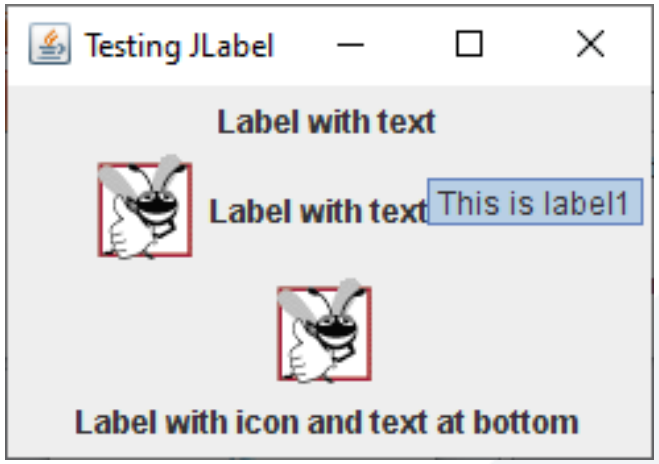
```
package ch12GUI;
//Fig. 12.6: LabelFrame.java    JLabels with text and icons.
import java.awt.FlowLayout; // specifies how components are arranged
import javax.swing.JFrame;  // provides basic window features
import javax.swing.JLabel;  // displays text and/or images
import javax.swing.SwingConstants; // common constants used with Swing
import javax.swing.Icon;    // interface used to manipulate images
import javax.swing.ImageIcon;       // loads images
public class LabelFrame extends JFrame
{private final JLabel label1; // JLabel constructed with just text
 private final JLabel label2; // JLabel constructed with text and icon
 private final JLabel label3; // JLabel with added text and icon
// LabelFrame constructor adds JLabels to JFrame
public LabelFrame()
{   super("Testing JLabel");
    setLayout(new FlowLayout());    // set frame layout
```

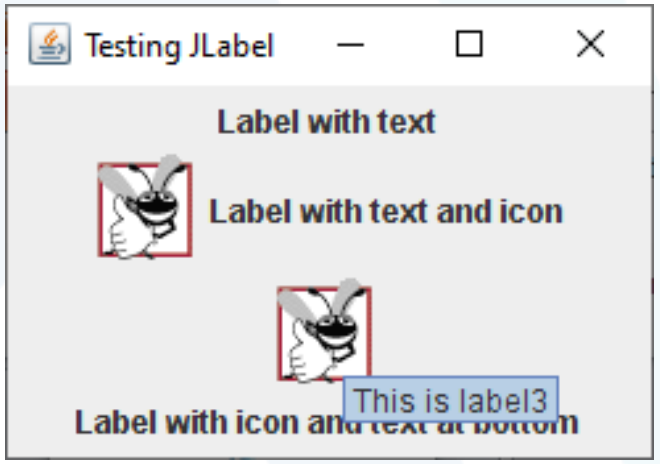
12.5 Displaying Text and Images Using Labels

```
        // JLabel constructor with a string argument
        label1 = new JLabel("Label with text");
        label1.setToolTipText("This is label1");
        add(label1); // add label1 to JFrame
        // JLabel constructor with string, Icon and alignment arguments
        Icon bug = new ImageIcon(getClass().getResource("bug1.png"));
        label2 = new JLabel("Label with text and icon", bug, SwingConstants.LEFT);
        label2.setToolTipText("This is label2");
        add(label2); // add label2 to JFrame
        label3 = new JLabel(); // JLabel constructor no arguments
        label3.setText("Label with icon and text at bottom");
        label3.setIcon(bug); // add icon to JLabel
        label3.setHorizontalTextPosition(SwingConstants.CENTER);
        label3.setVerticalTextPosition(SwingConstants.BOTTOM);
        label3.setToolTipText("This is label3");
        add(label3); // add label3 to JFrame
    }
} // end class LabelFrame
```

12.5 Displaying Text and Images Using Labels

```
 package ch12GUI;
//Fig. 12.7: LabelTest.java   Testing LabelFrame.
import javax.swing.JFrame;
public class LabelTest
{public static void main(String[] args)
{   LabelFrame labelFrame = new LabelFrame();
    labelFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    labelFrame.setSize(260, 180);
    labelFrame.setVisible(true); }
} // end class LabelTest
```





- تحميل مصدر الصورة في السطر، يستدعي التعبير getClass().getResource("bug1.png") المنهج getClass() (الموروثة بشكل غير مباشر من الصنف Object) لاسترداد مرجع إلى كائن Class الذي يمثل تعريف صنف LabelFrame. يتم بعد ذلك استخدام هذا المرجع لاستدعاء getResource التابع لمنهج Class، والذي يُرجع موقع الصورة كعنوان URL. يستخدم منشئ ImageIcon عنوان URL لتحديد موقع الصورة، ثم يقوم بتحميلها .
- تقوم JVM بتحميل تصريحات الأصناف في الذاكرة باستخدام مُحمل الصنف. يعرف مُحمل الصنف مكان وجود كل صنف يقوم بتحميلها على القرص. تستخدم طريقة getResource مُحمل صنف كائن الصنف لتحديد موقع المورد، مثل ملف صورة. في هذا المثال، يتم تخزين ملف الصورة في نفس الموقع مثل ملف LabelFrame.class. تعمل التقنيات الموضحة هنا على تمكين التطبيق من تحميل ملفات الصور من المواقع المرتبطة بموقع ملف الصنف .
- تحميل واجهة موارد الصور SwingConstants (packagejavax.swing) تعلن عن مجموعة من الثوابت الصحيحة الشائعة (مثل SwingConstants.LEFT وSwingConstants.CENTER وSwingConstants.RIGHT) التي يتم استخدامها مع العديد من مكونات Swing. افتراضيًا، يظهر النص على يمين الصورة عندما تحتوي التسمية على نص وصورة معًا. يمكن ضبط المحاذاة الأفقية والرأسية لـ JLabel باستخدام الأساليب setHorizontalAlignment وsetVerticalAlignment، على التوالي. يحدد السطر نص تلميح الأداة للتسمية 2، ويضيف السطر التسمية 2 إلى JFrame.

يتم قراءة البيانات من لوحة المفاتيح بصيغة أسكي إذا كانت القراءة بايت واحد وبصيغة سلسلة إذا كانت القراءة بعدة بايتات وعلى المبرمج تحويل من سلسلة نصية إلى أرقام عددية int باستخدام الدوال الخاصة بالسلاسل للتعامل معها كأعداد

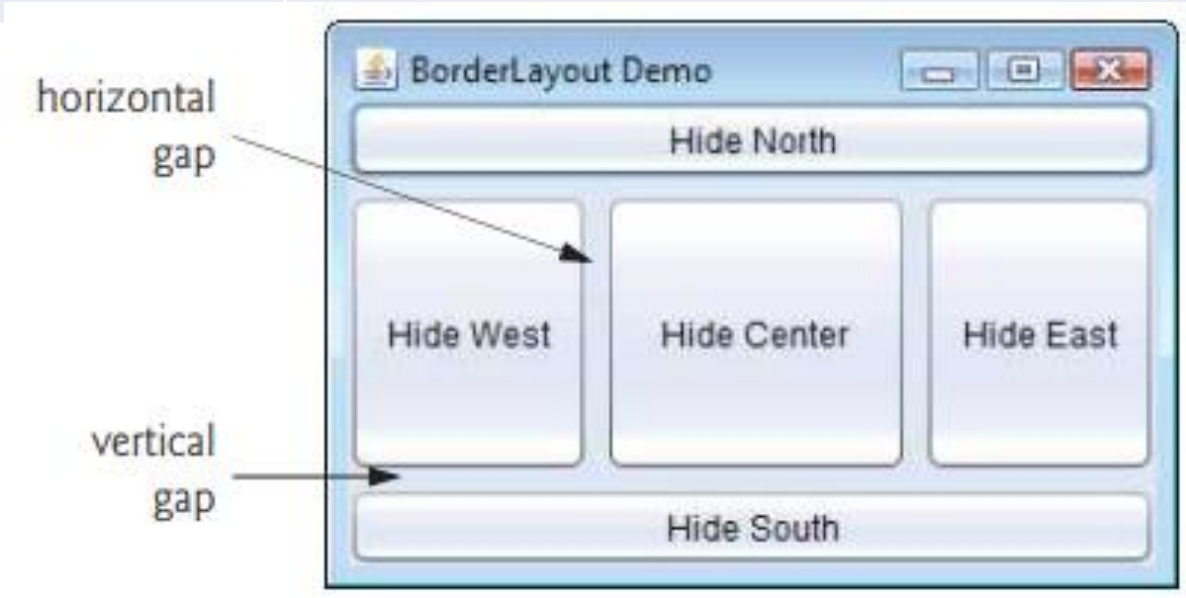
```
package Rectangle7_10_2023;
import java.io.*;
public class Excepy {
    public static void main(String args[])throws IOException
      {// American Standard Code for Information Interchange
 int b; b=System.in.read();
//for(int i='a'; i<='z';i++)
System.out.println("ASCII "+b);
   }
 }
```

Layout Managers 1

- Absolute positioning
  - By setting a Container's layout to null, setLayout(null)
  - setBounds(x, y, w, h)
- Layout managers
  - Available for arranging GUI components
  - Processes layout details
  - Programmer can concentrate on basic "look and feel"
  - Interface `LayoutManager`
- Visual programming in an IDE

Layout Managers 2

Layout manager	Description
FlowLayout	Default for java.awt.Applet, java.awt.Panel and javax.swing.JPanel. Places components sequentially (left to right) in the order they were added. It is also possible to specify the order of the components by using the Container method add, which takes a Component and an integer index position as arguments.
BorderLayout	Default for the content panes of JFrames (and other windows) and JApplets. Arranges the components into five areas: NORTH, SOUTH, EAST, WEST and CENTER.
GridLayout	Arranges the components into rows and columns.



Layout Managers

- `FlowLayout`
  - Most basic layout manager
  - GUI components placed in container from left to right
- `BorderLayout`
  - Arranges components into five regions
    - `NORTH` (top of container) `SOUTH` (bottom of container)
    - `EAST` (right side) `WEST` (left side)
    - `CENTER` (center of container)
- `GridLayout`
  - Divides container into grid of specified row an columns
  - Components are added starting at top-left cell
    - Proceed left-to-right until row is full

13.4 Event Handling

- GUIs are *event driven*
  - Generate *events* when user interacts with GUI
    - e.g., moving mouse, pressing button, typing in text field, etc.
    - Class `java.awt.AWTEvent`

package java.awt.event

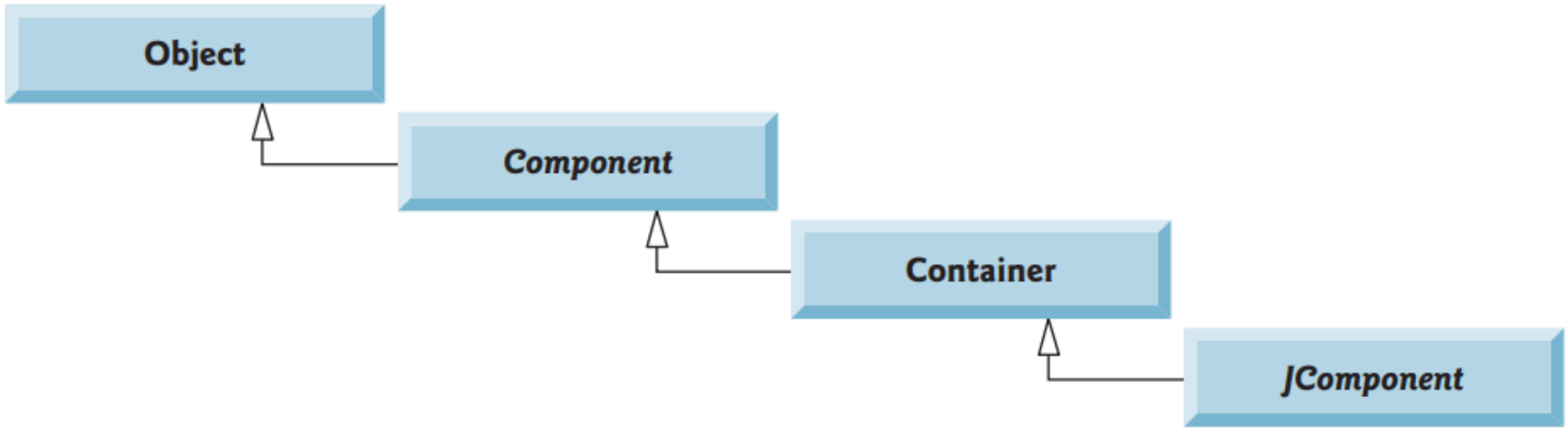


Fig. 12.5 | Common superclasses of the lightweight Swing components.

package java.awt.event

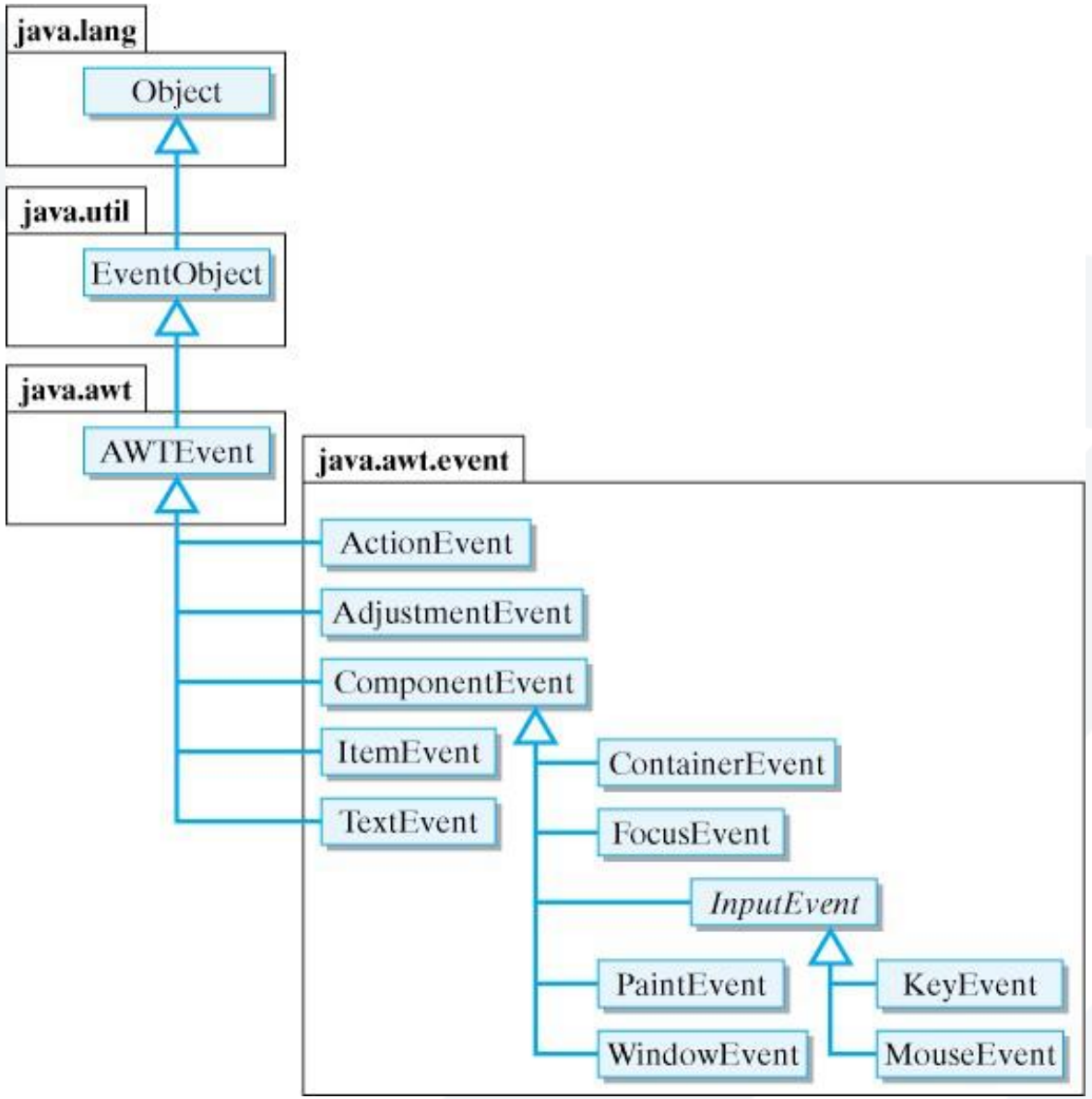


Fig 12.11Some event classes of package java.awt.event

13.4 Event Handling

- Event-handling model
  - Three parts
    - Event source
      - GUI component with which user interacts
    - Event object
      - Encapsulates information about event that occurred
    - Event listener
      - Receives event object when notified, then responds
  - Programmer must perform two tasks
    - Register event listener for event source
    - Implement event-handling method (event handler)

java.awt.event

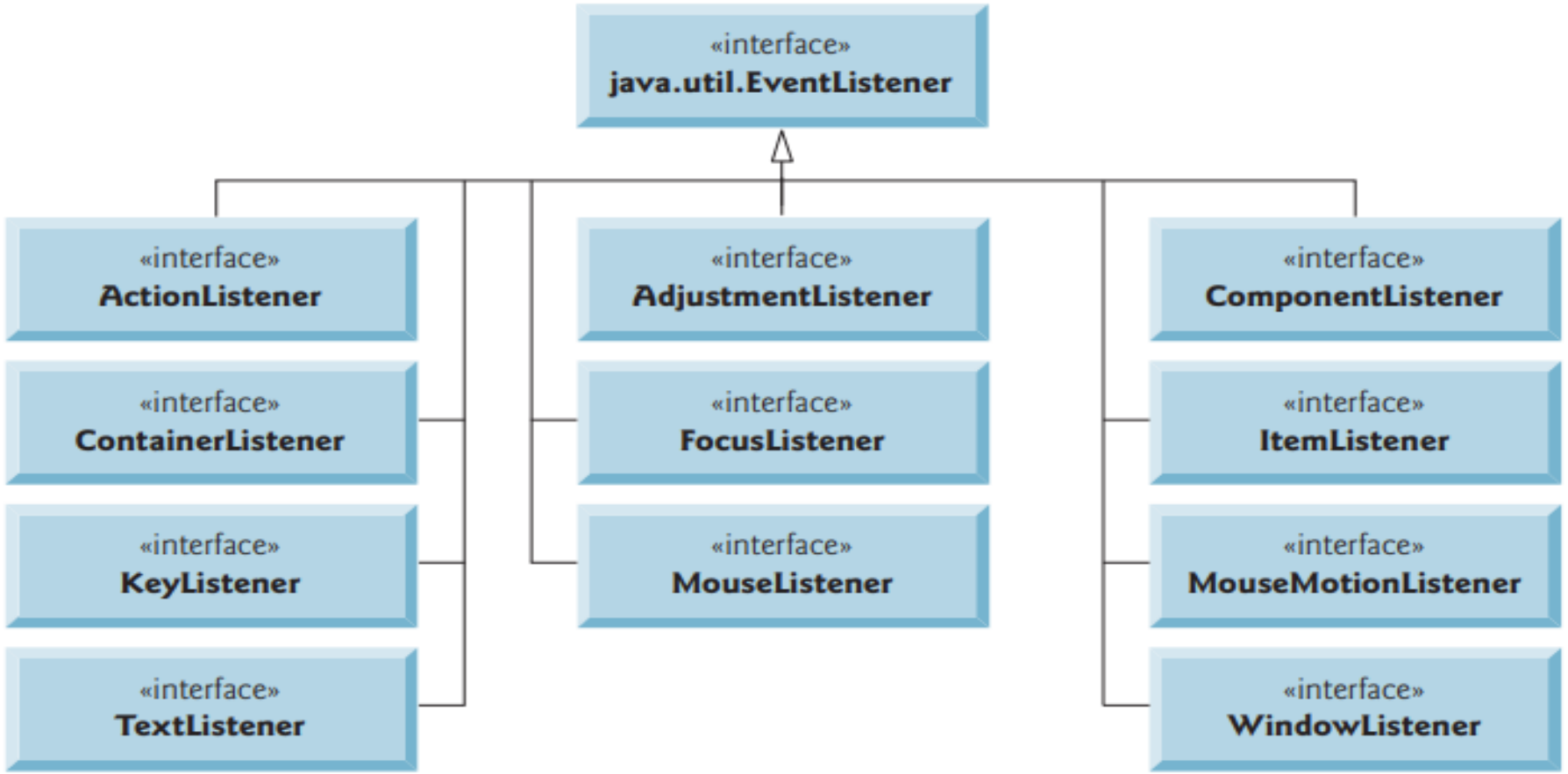


Fig. 12.12 | Some common event-listener interfaces of package `java.awt.event`.

13.5 TextFields

- `JTextField`
  - Single-line area in which user can enter text
- `JPasswordField`
  - Extends `JTextField`
  - Hides characters that user enters
- Event-handling model

13.5 TextFields

```
package ch12GUI;

//Fig. 12.9: TextFieldFrame.java
// JTextFields and JPasswordFields.
 import java.awt.FlowLayout;
 import java.awt.event.ActionListener;
 import java.awt.event.ActionEvent;
 import javax.swing.JFrame;
 import javax.swing.JTextField;
 import javax.swing.JPasswordField;
 import javax.swing.JOptionPane;
 public class TextFieldFrame extends JFrame
 { private final JTextField textField1; // text field with set size
 private final JTextField textField2; // text field with text
 private final JTextField textField3; // text field with text and size
 private final JPasswordField passwordField; // password field with text
 // TextFieldFrame constructor adds JTextFields to JFrame
 public TextFieldFrame()
 { super("Testing JTextField and JPasswordField");
 setLayout(new FlowLayout());
//construct text field with 10 columns
 textField1 = new JTextField(10);
 add(textField1); // add textField1 to JFrame
//construct text field with default text
 textField2 = new JTextField("Enter text here");
  add(textField2); // add textField2 to JFrame
```

Declare three `JTextField`s and one `JPasswordField`

First `JTextField` contains empty string

Second `JTextField` contains text "`Enter text here`"

13.5 TextFields

```
//construct text field with default text and 21 columns
textField3 = new JTextField("Uneditable text field", 21);
textField3.setEditable(false); // disable editing
add(textField3); // add textField3 to JFrame
//construct password field with default text
passwordField = new JPasswordField("Hidden text");
add(passwordField); // add passwordField to JFrame
//register event handlers
TextFieldHandler handler = new TextFieldHandler();
textField1.addActionListener(handler);
textField2.addActionListener(handler);
textField3.addActionListener(handler);
passwordField.addActionListener(handler);
}
//private inner class for event handling
private class TextFieldHandler implements ActionListener
{  // process text field events
 @Override
public void actionPerformed(ActionEvent event)
 {  String string = "";
  // user pressed Enter in JTextField textField1
if (event.getSource() == textField1)
string = String.format("textField1: %s", event.getActionCommand());
```

Third `JTextField` contains uneditable text

`JPasswordField` contains text "`Hidden text`," but text appears as series of asterisks (*)

Register GUI components with `TextFieldHandler` (register for `ActionEvents`)

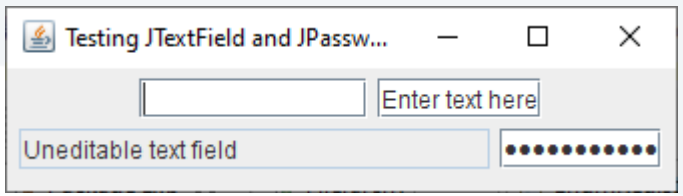
Every `TextFieldHandler` instance is an `ActionListener`

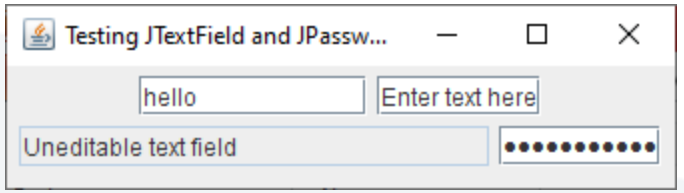
Method `actionPerformed` invoked when user presses Enter in GUI field

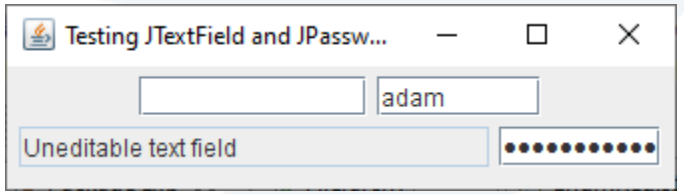
13.5 TextFields

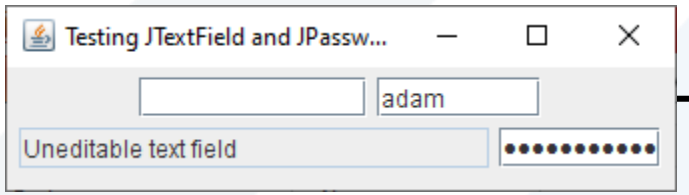
```
// user pressed Enter in JTextField textField2
  else if (event.getSource() == textField2)
 string = String.format("textField2: %s", event.getActionCommand());
// user pressed Enter in JTextField textField3
  else if (event.getSource() == textField3)
 string = String.format("textField3: %s", event.getActionCommand());
// user pressed Enter in JTextField passwordField
  else if (event.getSource() == passwordField)
  string = String.format("passwordField: %s",  event.getActionCommand());
// display JTextField content
  JOptionPane.showMessageDialog(null, string);    }
  } // end private inner class TextFieldHandler
 } // end class TextFieldFrame

//Fig. 12.10: TextFieldTest.java Testing TextFieldFrame.
 import javax.swing.JFrame;
 public class TextFieldTest
 { public static void main(String[] args)
 { TextFieldFrame textFieldFrame = new TextFieldFrame();
 textFieldFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 textFieldFrame.setSize(350, 100);
 textFieldFrame.setVisible(true);
 }    } // end class TextFieldTest
```

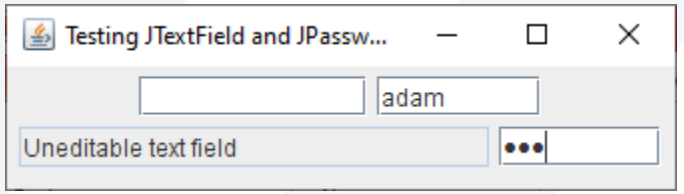
13.5 TextFields











13.6 How Event Handling Works

- يتم تسجيل معالج الحدث.
- من خلال طريقة المكون addActionListenerالمضافه على المكونات المراد تسجيل الحدث عنها.
  - يتم إرسال الحدث إلى المستمعين من النوع المناسب فقط .
  - يتم إ ستدعاء الإجراء المناسب للحدث.
  - يحتوي كل نوع حدث على واجهة مستمع الحدث المقابلة.

- يحتوي كائن الحدث على معرف الحدث نوع الحدث ومصدره وبارامترات اخرى.

Event registration

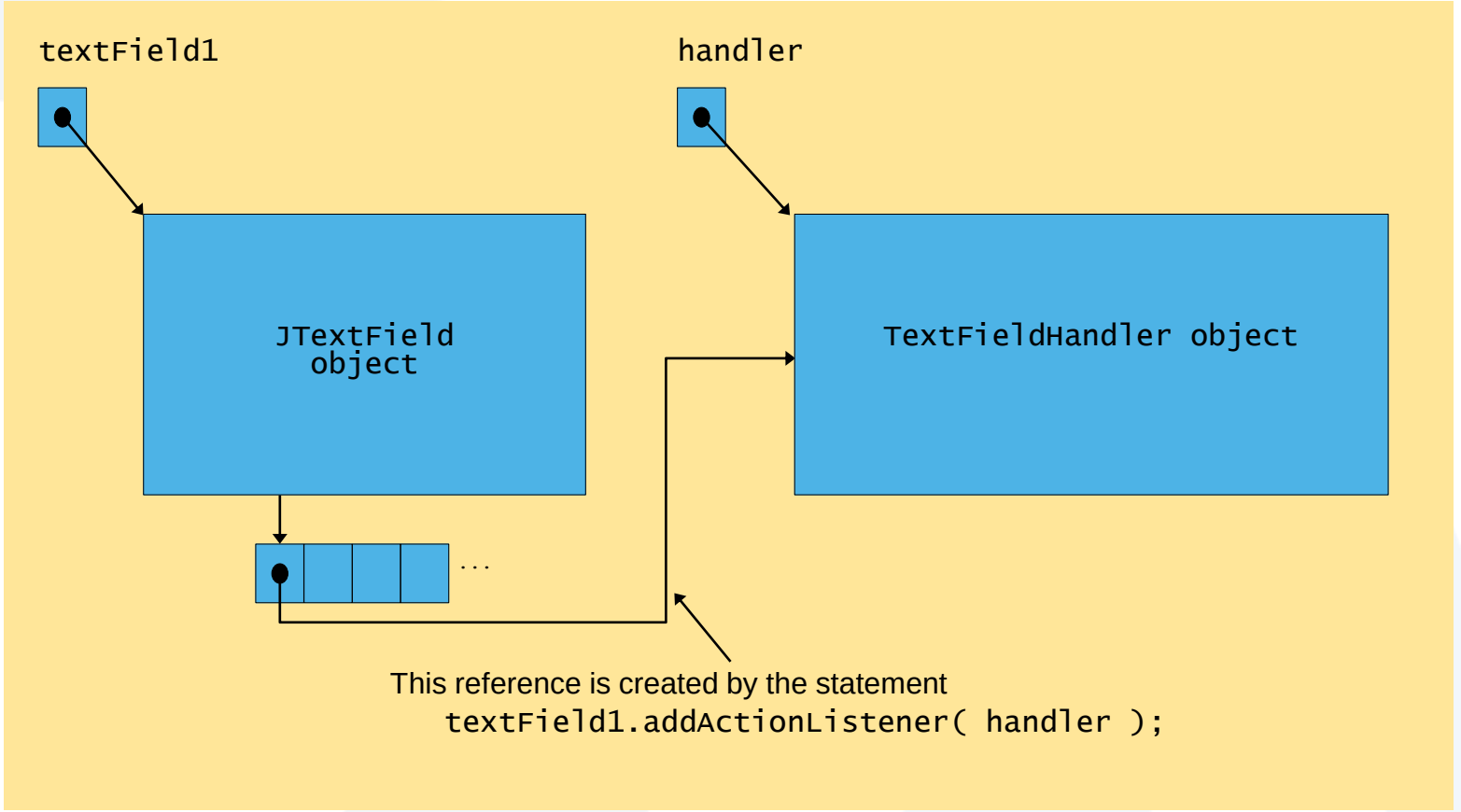


Fig. 13.8 Event registration for `JTextField textField1`

13.7 `JButton`

- Button
  - Component user clicks to trigger a specific action
  - Several different types
    - Command buttons
    - Check boxes
    - Toggle buttons
    - Radio buttons
  - `javax.swing.AbstractButton` subclasses
    - Command buttons are created with class `JButton`
      - Generate `ActionEvent`s when user clicks button

Swing button hierarchy

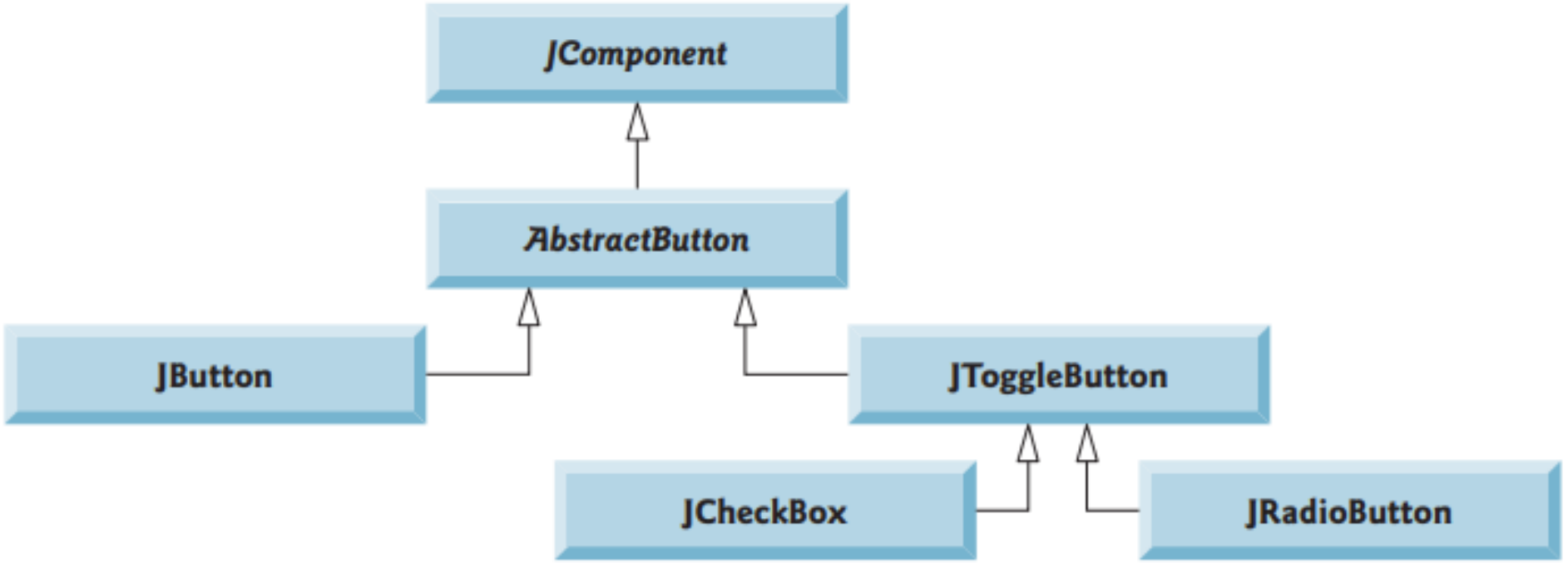


Fig. 12.14 | Swing button hierarchy.

Swing button hierarchy

```
import java.awt.FlowLayout;
 import java.awt.event.ActionListener;
 import java.awt.event.ActionEvent;
 import javax.swing.JFrame;
 import javax.swing.JButton;
 import javax.swing.Icon;
 import javax.swing.ImageIcon;
 import javax.swing.JOptionPane;
public class ButtonFrame extends JFrame
 { private final JButton plainJButton; // button with just text
   private final JButton fancyJButton; // button with icons
  // ButtonFrame adds JButtons to JFrame
 public ButtonFrame()  { super("Testing Buttons");  setLayout(new FlowLayout());
 plainJButton = new JButton("Plain Button"); // button with text
 add(plainJButton); // add plainJButton to JFrame
Icon bug1 = new ImageIcon(getClass().getResource("bug1.png"));
Icon bug2 = new ImageIcon(getClass().getResource("bug2.png"));
fancyJButton = new JButton("Fancy Button", bug1); // set image
fancyJButton.setRolloverIcon(bug2); // set rollover image
add(fancyJButton); // add fancyJButton to JFrame
```

Create two references to `JButton` instances

Instantiate `JButton` with text

Instantiate `JButton` with image and *rollover* image

Swing button hierarchy

```
 // create new ButtonHandler for button event handling
ButtonHandler handler = new ButtonHandler();
fancyJButton.addActionListener(handler);
plainJButton.addActionListener(handler); }
 // inner class for button event handling
 private class ButtonHandler implements ActionListener
 { // handle button event
 @Override
 public void actionPerformed(ActionEvent event)
 { JOptionPane.showMessageDialog(ButtonFrame.this , String.format("You pressed:
%s",event.getActionCommand() )); }
   }
 } // end class ButtonFrame
  /* Accessing the this Reference in an Object of a Top-Level Class from an Inner Class When you
  *  execute this application and click one of its buttons, notice that the message dialog that
  * appears is centered over the application's window. This occurs because the call to JOptionPane
  *   method showMessageDialog uses ButtonFrame.this rather than null as the first argument.  */

import javax.swing.JFrame;
 public class ButtonTest    {
```

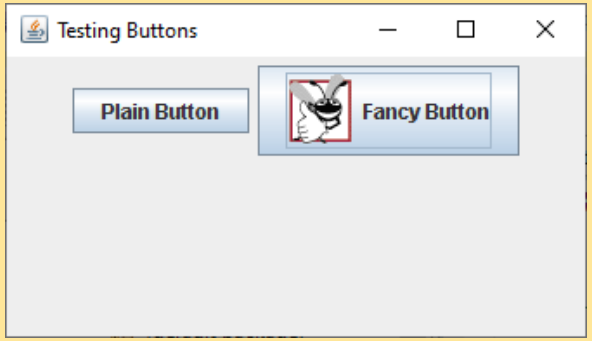
Instantiate `ButtonHandler` for `JButton` event handling

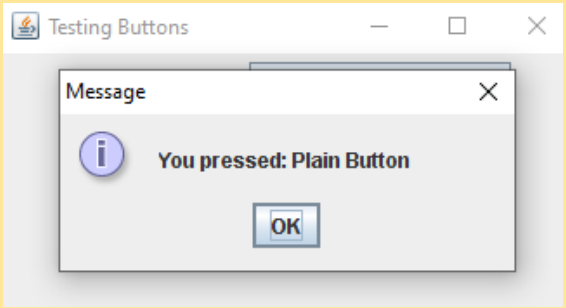
Register `JButton`s to receive events from `ButtonHandler`

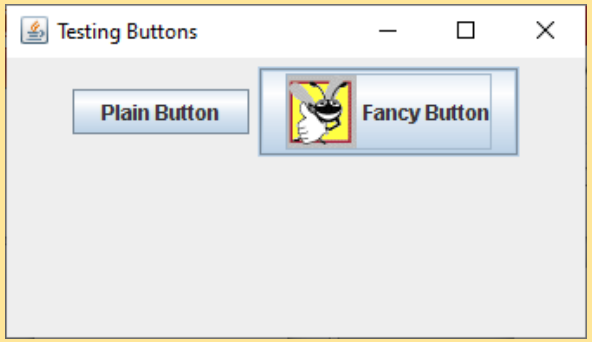
When user clicks `JButton`, `ButtonHandler` invokes method `actionPerformed` of all registered listeners

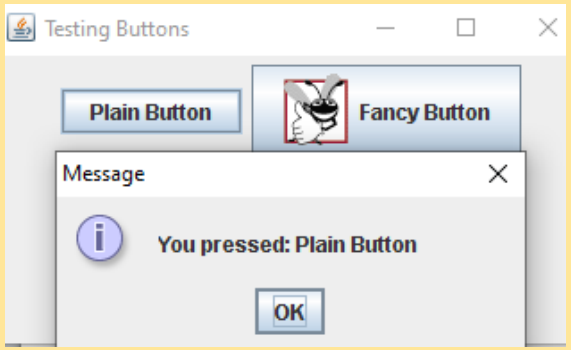
Swing button hierarchy

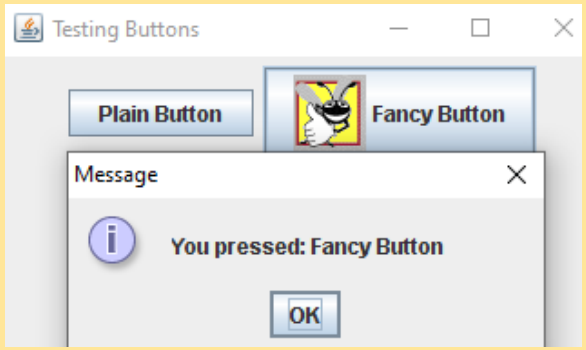
```
public static void main(String[] args)
 {ButtonFrame buttonFrame = new ButtonFrame();
 buttonFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 buttonFrame.setSize(350, 200);
 buttonFrame.setVisible(true);    }
 } // end class ButtonTest
```











13.8 JCheckBox and JRadioButton

- State buttons
  - On/Off or `true`/`false` values
  - Java provides three types
    - `JToggleButton`
    - `JCheckBox`
    - `JRadioButton`

CheckBoxTest.java 1

```
//Fig. 12.17: CheckBoxFrame.java  JCheckBoxes and item events.
 import java.awt.FlowLayout;
 import java.awt.Font;
 import java.awt.event.ItemListener;
 import java.awt.event.ItemEvent;
 import javax.swing.JFrame;
 import javax.swing.JTextField;
 import javax.swing.JCheckBox;
 public class CheckBoxFrame extends JFrame  {
private final JTextField textField;      // displays text in changing fonts
private final JCheckBox boldJCheckBox;   // to select/deselect bold
private final JCheckBox italicJCheckBox; // to select/deselect italic
  // CheckBoxFrame constructor adds JCheckBoxes to JFrame
 public CheckBoxFrame() { super("JCheckBox Test");
 setLayout(new FlowLayout());  // set up JTextField and set its font
 textField = new JTextField("Watch the font style change", 20);
 textField.setFont(new Font("Serif", Font.PLAIN, 14));
 add(textField); // add textField to JFrame
```

Declare two `JCheckBox` instances

Set `JTextField` font to Serif, 14-point plain

CheckBoxTest.java 2

```
 boldJCheckBox = new JCheckBox("Bold");
 italicJCheckBox = new JCheckBox("Italic");
 add(boldJCheckBox); // add bold checkbox to JFrame
 add(italicJCheckBox); // add italic checkbox to Jframe  register listeners for JCheckBoxes
CheckBoxHandler handler = new CheckBoxHandler();
boldJCheckBox.addItemListener(handler);
italicJCheckBox.addItemListener(handler);   }
 // private inner class for ItemListener event handling
 private class CheckBoxHandler implements ItemListener
 {// respond to checkbox events
 @Override
 public void itemStateChanged(ItemEvent event)
 { Font font = null; // stores the new Font
 // determine which CheckBoxes are checked and create Font
 if (boldJCheckBox.isSelected() && italicJCheckBox.isSelected() )
 font = new Font("Serif", Font.BOLD + Font.ITALIC, 14);
 else if (boldJCheckBox.isSelected()) font = new Font("Serif", Font.BOLD, 14);
```

Instantiate `JCheckBox`s for bolding and italicizing `JTextField` text, respectively

Register `JCheckBox`s to receive events from `CheckBoxHandler`

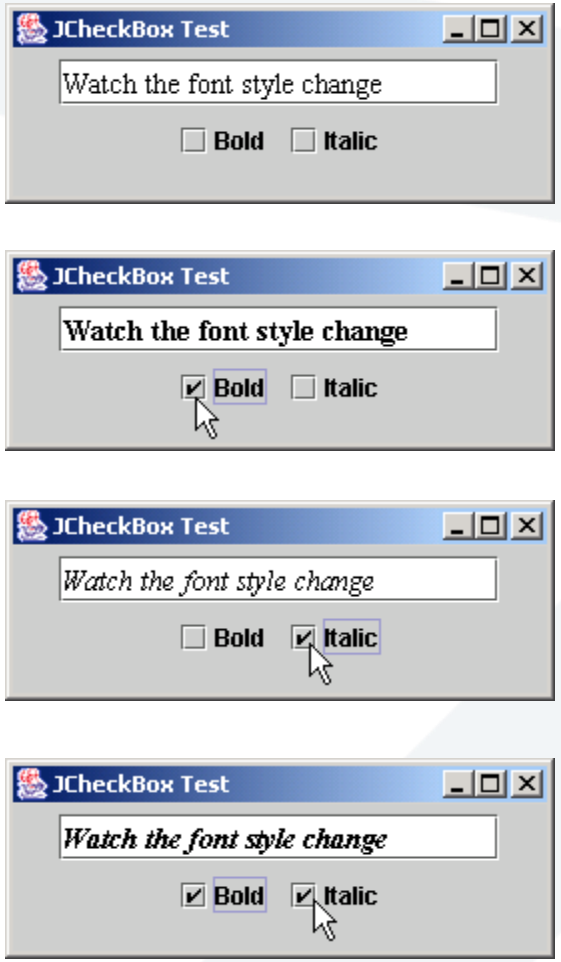
When user selects `JCheckBox`, `CheckBoxHandler` invokes method `itemStateChanges` of all registered listeners

CheckBoxTest.java 3

```
else if (italicJCheckBox.isSelected())
 font = new Font("Serif", Font.ITALIC, 14);
 else    font = new Font("Serif", Font.PLAIN, 14);
 textField.setFont(font);
        }       }
 } // end class CheckBoxFrame

// Fig. 12.18: CheckBoxTest.java Testing CheckBoxFrame.
 import javax.swing.JFrame;
 public class CheckBoxTest {
 public static void main(String[] args)
 {  CheckBoxFrame checkBoxFrame = new CheckBoxFrame();
 checkBoxFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 checkBoxFrame.setSize(275, 100);
 checkBoxFrame.setVisible(true);
    }
 } // end class CheckBoxTest
```

Change `JTextField` font, depending on which `JCheckBox` was selected

CheckBoxTest



CheckBoxTest.java

RadioButtonTest 1

```
//Fig. 12.19: RadioButtonFrame.java
 // Creating radio buttons using ButtonGroup and JRadioButton.
 import java.awt.FlowLayout;
 import java.awt.Font;
 import java.awt.event.ItemListener;
 import java.awt.event.ItemEvent;
 import javax.swing.JFrame;
 import javax.swing.JTextField;
 import javax.swing.JRadioButton;
 import javax.swing.ButtonGroup;
 public class RadioButtonFrame extends JFrame
 { private final JTextField textField; // used to display font changes
 private final Font plainFont; // font for plain text
 private final Font boldFont; // font for bold text
 private final Font italicFont; // font for italic text
 private final Font boldItalicFont; // font for bold and italic text
 private final JRadioButton plainJRadioButton; // selects plain text
 private final JRadioButton boldJRadioButton; // selects bold text
```

Declare two `JRadioButton` instances

RadioButtonTest 2

```
private final JRadioButton italicJRadioButton; // selects italic text
 private final JRadioButton boldItalicJRadioButton; // bold and italic

private final ButtonGroup radioGroup;
// holds radio buttons
 // RadioButtonFrame constructor adds JRadioButtons to JFrame
 public RadioButtonFrame() {
 super("RadioButton Test");
 setLayout(new FlowLayout());
 textField = new JTextField("Watch the font style change", 25);
 add(textField); // add textField to JFrame
 // create radio buttons

plainJRadioButton = new JRadioButton("Plain", true);
boldJRadioButton = new JRadioButton("Bold", false);
italicJRadioButton = new JRadioButton("Italic", false);
boldItalicJRadioButton = new JRadioButton("Bold/Italic", false);
```

Declare two `JRadioButton` instances

`JRadioButton`s normally appear as a `ButtonGroup`

Instantiate `JRadioButton`s for manipulating `JTextField`

RadioButtonTest 3

```
//create logical relationship between JRadioButtons
radioGroup = new ButtonGroup();           // create ButtonGroup
radioGroup.add(plainJRadioButton);        // add plain to group
radioGroup.add(boldJRadioButton);         // add bold to group
radioGroup.add(italicJRadioButton);       // add italic to group
radioGroup.add(boldItalicJRadioButton); // add bold and italic
 // create font objects
 plainFont = new Font("Serif", Font.PLAIN, 14);
 boldFont = new Font("Serif", Font.BOLD, 14);
 italicFont = new Font("Serif", Font.ITALIC, 14);
 boldItalicFont = new Font("Serif", Font.BOLD + Font.ITALIC, 14);
 textField.setFont(plainFont);
//register events for JRadioButtons
plainJRadioButton.addItemListener(new RadioButtonHandler(plainFont));
boldJRadioButton.addItemListener( new RadioButtonHandler(boldFont));
italicJRadioButton.addItemListener(new RadioButtonHandler(italicFont));
boldItalicJRadioButton.addItemListener(new RadioButtonHandler(boldItalicFont));
 }
```

JRadioButtons belong to ButtonGroup

Register JRadioButtons to receive events from RadioButtonHandler

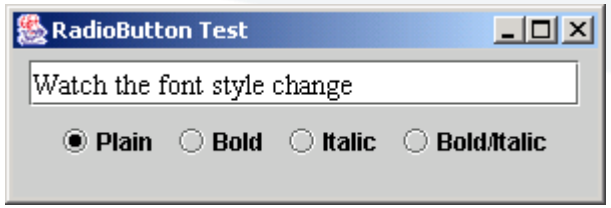
RadioButtonTest 4

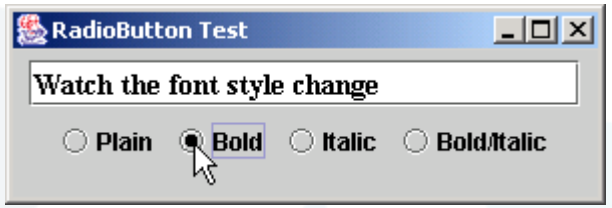
```
// private inner class to handle radio button events
 private class RadioButtonHandler implements ItemListener
 {  private Font font; // font associated with this listener
 public RadioButtonHandler(Font f) { font = f;  }
  // handle radio button events
 @Override
 public void itemStateChanged(ItemEvent event) { textField.setFont(font);  }
 }   } // end class RadioButtonFrame

// Fig. 12.20: RadioButtonTest.java  Testing RadioButtonFrame.
 import javax.swing.JFrame;
 public class RadioButtonTest
 { public static void main(String[] args)
 { RadioButtonFrame radioButtonFrame = new RadioButtonFrame();
 radioButtonFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 radioButtonFrame.setSize(300, 100);
 radioButtonFrame.setVisible(true);  }
 } // end class RadioButtonTest
```

When user selects `JRadioButton`, `RadioButtonHandler` invokes method `itemStateChanged` of all registered listeners

Set font corresponding to `JRadioButton` selected

RadioButtonTest 4





ComboBoxTest 1

```
//Fig. 12.21: ComboBoxFrame.java. sometimes called a drop-down list
 // JComboBox that displays a list of image names
 import java.awt.FlowLayout;
 import java.awt.event.ItemListener;
 import java.awt.event.ItemEvent;
 import javax.swing.JFrame;
 import javax.swing.JLabel;
 import javax.swing.JComboBox;
 import javax.swing.Icon;
 import javax.swing.ImageIcon;
 public class ComboBoxFrame extends JFrame
 { private final JComboBox imagesJComboBox; // holds icon names
 private final JLabel label; // displays selected icon
 private static final String[] names =
 {"bug1.png", "bug2.png", "travelbug.png", "buganim.gif"};
```

ComboBoxTest 1

```
private final Icon[] icons = {
 new ImageIcon(getClass().getResource(names[0])),
 new ImageIcon(getClass().getResource(names[1])),
 new ImageIcon(getClass().getResource(names[2])),
 new ImageIcon(getClass().getResource(names[3]))};
 // ComboBoxFrame constructor adds JComboBox to JFrame
 public ComboBoxFrame()   { super("Testing JComboBox");
 setLayout(new FlowLayout()); // set frame layout
 imagesJComboBox = new JComboBox(names); // set up JComboBox
 imagesJComboBox.setMaximumRowCount(3); // display three rows
 imagesJComboBox.addItemListener( new ItemListener() // anonymous inner class
 { // handle JComboBox event
 @Override
 public void itemStateChanged(ItemEvent event) {
 // determine whether item selected
```

Instantiate `JComboBox` to show three `String`s from `names` array at a time

Register `JComboBox` to receive events from anonymous `ItemListener`

When user selects item in `JComboBox`, `ItemListener` invokes method `itemStateChanged` of all registered listeners

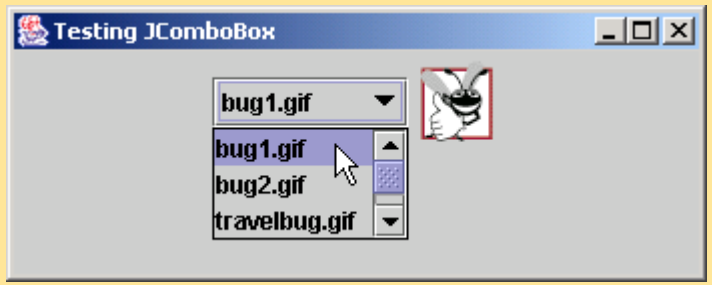
ComboBoxTest 1

```
if (event.getStateChange() == ItemEvent.SELECTED)
 label.setIcon(icons[imagesJComboBox.getSelectedIndex()]);
 }
 } // end anonymous inner class
); // end call to addItemListener
 add(imagesJComboBox); // add combobox to JFrame
 label = new JLabel(icons[0]); // display first icon
 add(label); // add label to JFrame
 }
 } // end class ComboBoxFrame
// Fig. 12.22: ComboBoxTest.java
 // Testing ComboBoxFrame.
 import javax.swing.JFrame;
 public class ComboBoxTest
 { public static void main(String[] args)
```

Set appropriate `Icon` depending on user selection

ComboBoxTest 1

```
ComboBoxFrame comboBoxFrame = new ComboBoxFrame();
 comboBoxFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 comboBoxFrame.setSize(350, 150);
 comboBoxFrame.setVisible(true);
 }
 } // end class ComboBoxTest
```



انتهت محاضرة الأسبوع السابع والثامن

تمرين امتحان سابق

```
import java.awt.FlowLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JOptionPane;
import javax.swing.JLabel;
import javax.swing.JButton;

public class TFieldFraFin extends JFrame
        {private final JLabel label1;                    private final JLabel label2;
        private final JTextField textField1;             private final JTextField textField2;
        private final JButton JBSum;
public TFieldFraFin()
        {super("Testing sum 2 number");        setLayout(new FlowLayout());
        label1 = new JLabel("Enter first num");          add(label1);
        textField1 = new JTextField(5);                  add(textField1);
        label2 = new JLabel("Enter second num");         add(label2);
        textField2 = new JTextField(5);                  add(textField2);
        JBSum = new JButton("JBSum");                    add(JBSum);
```

تمرين امتحان سابق

```
//register event handlers
ButtonHandler handler = new ButtonHandler(); JBSum.addActionListener(handler);     }
private class ButtonHandler implements ActionListener
{  // process text field events
  @Override
 public void actionPerformed(ActionEvent event)
        {       int a = Integer.parseInt(textField1.getText()) ;
        int b = Integer.parseInt(textField2.getText()) ;     int c=a+b;
        textField1.setText("");        String string = "the sum = "+c;System.out.println(c);
        JOptionPane.showMessageDialog(null, string);
        }
}       } // end class TFieldFraFin

import javax.swing.JFrame;
public class TFieldFraFinTest
 {public static void main(String[] args){
TFieldFraFin textFieldFrame = new TFieldFraFin();
textFieldFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
textFieldFrame.setSize(250, 200);     textFieldFrame.setVisible(true);
   } } // end class TFieldFraFinTest
```