CSE 1325

Week of 11/28/2022

Instructor : Donna French

GUI

A GUI (graphical user interface) is built from GUI components.

These are sometimes called controls or widgets (window gadgets).

A GUI component is an object with which the user interacts via the mouse, keyboard or another form of input.

We are going to learn about Swing from the `javax.swing` package.

GUI

Many IDEs provide GUI design tools and the IDE will generate the GUI code for you.

Each IDE generates this code differently (and not very well).

We will write our own GUI code by hand and not rely on the IDE to create it for us.

GUI

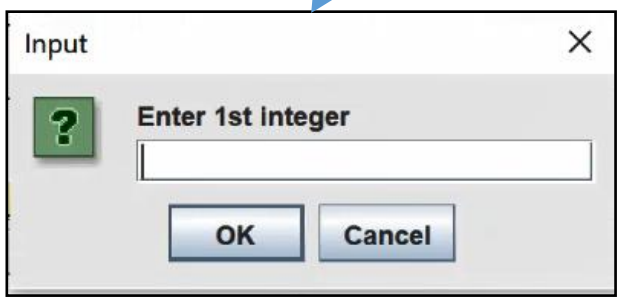
Most applications on a daily basis use windows or dialog boxes (called dialogs) to interact with the user.

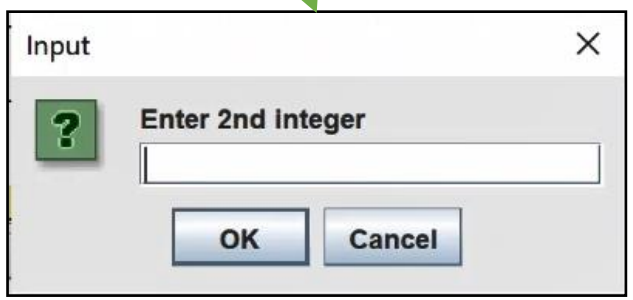
Dialog boxes are windows in which program display important messages to the user or obtain information from the user.

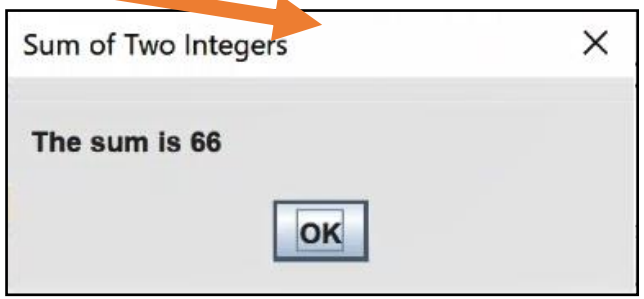
Java's `JOptionPane` class (package `javax.swing`) provides prebuilt dialog boxes for both input and output.

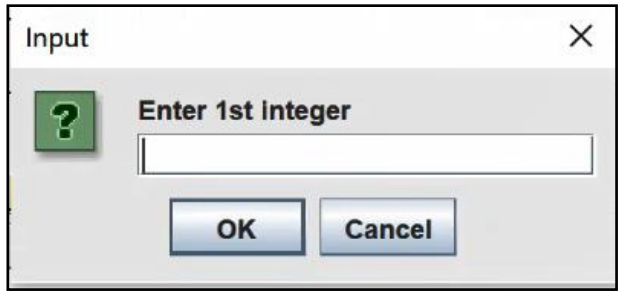
```java
public class GUIDemo
{
    public static void main(String[] args)
    {
        String firstNumber = JOptionPane.showInputDialog("Enter 1st integer ");
        String secondNumber = JOptionPane.showInputDialog("Enter 2nd integer ");

        int number1 = Integer.parseInt(firstNumber);
        int number2 = Integer.parseInt(secondNumber);

        int sum = number1 + number2;

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

```
public class GUIDemo
{
    public static void main(String[] args)
    {
        String firstNumber = JOptionPane.showInputDialog("Enter 1st integer ");
        String secondNumber = JOptionPane.showInputDialog("Enter 2nd integer ");

        int number1 = Integer.parseInt(firstNumber);
        int number2 = Integer.parseInt(secondNumber);

        int sum = number1 + number2;

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

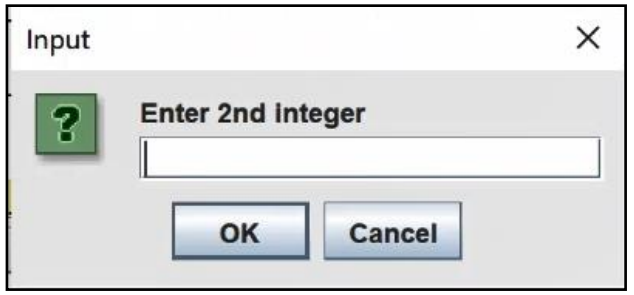








GUI - Dialog




```
String firstNumber = JOptionPane.showInputDialog("Enter 1st integer ");
String secondNumber = JOptionPane.showInputDialog("Enter 2nd integer ");
```

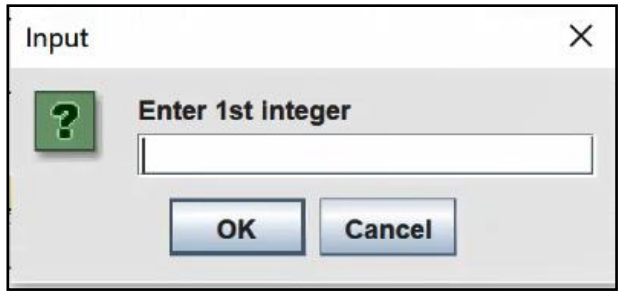
`showInputDialog` is a `static` method of `JOptionPane`

Displays an input dialog box using the passed in string as the prompt

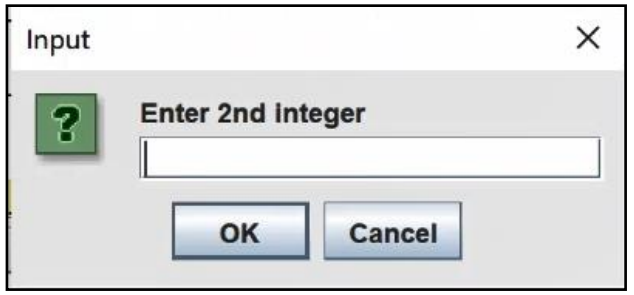
User types characters in the text field and clicks OK or presses ENTER to submit the string to the program

Clicking OK also dismisses (hides) the dialog

An input dialog can only input string (this is pretty typical of GUI components)



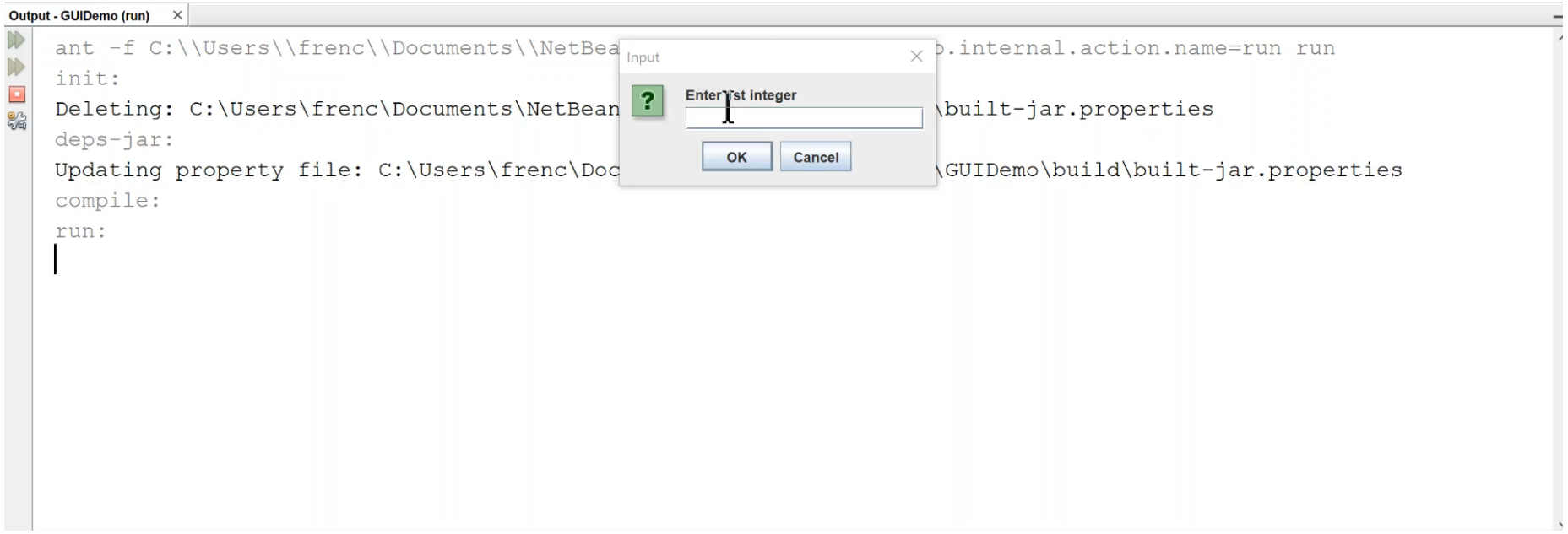
GUI - Dialog

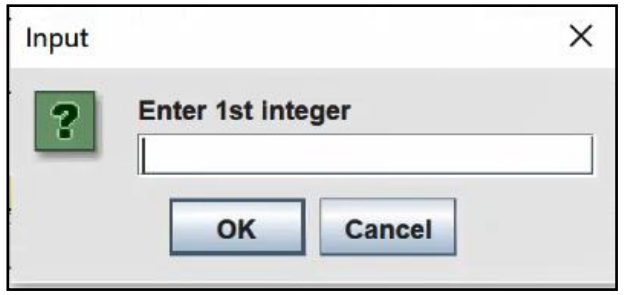



```
String firstNumber = JOptionPane.showInputDialog("Enter 1st integer ");
String secondNumber = JOptionPane.showInputDialog("Enter 2nd integer ");
```

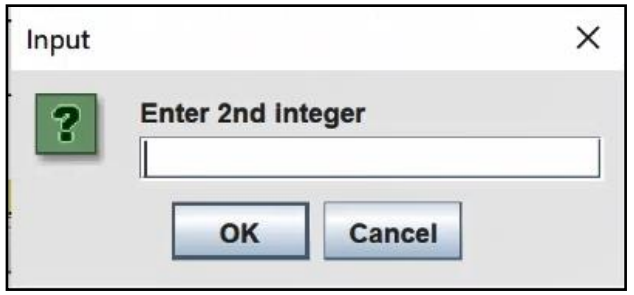
User can type any characters in the input dialog's text field

Our program is assuming that the user enters a valid integer.



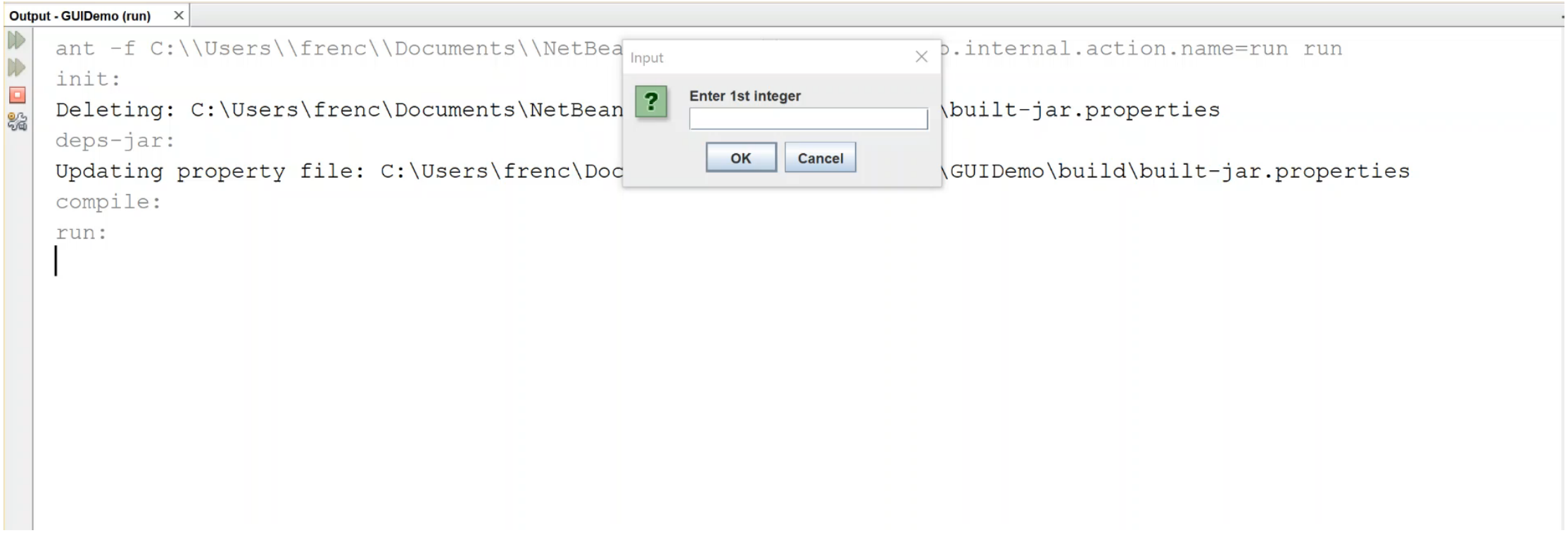


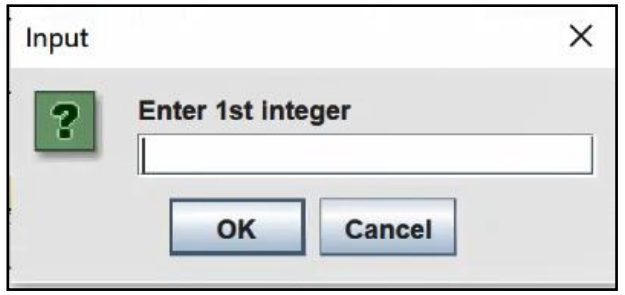
GUI - Dialog



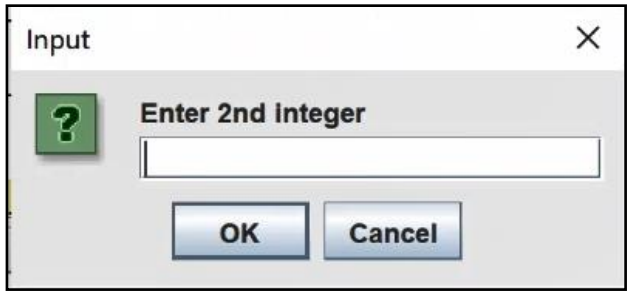

```
String firstNumber = JOptionPane.showInputDialog("Enter 1st integer ");
String secondNumber = JOptionPane.showInputDialog("Enter 2nd integer ");
```

If the user click `Cancel`, `showInputDialog()` returns `null`





GUI - Dialog



Each `JOptionPane` dialog that you display is a **modal** dialog

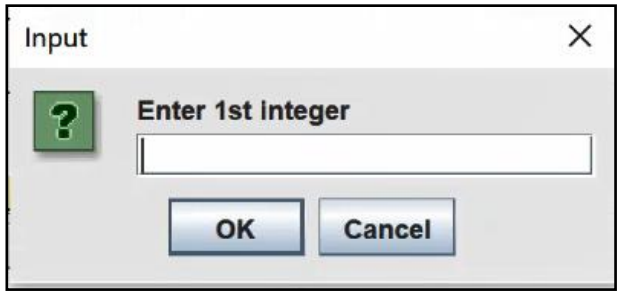
While the dialog is on the screen the user cannot interact with the rest of the application

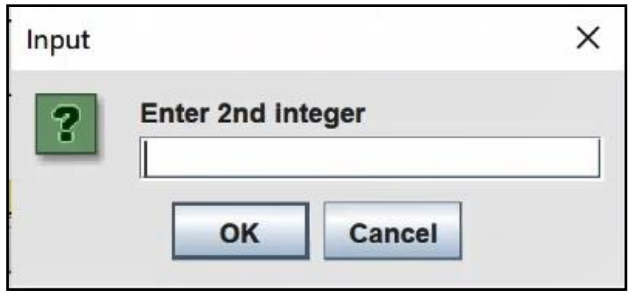
Be careful overusing **modal** dialogs

they can reduce the usability of your application

use modal dialog only when it's necessary to prevent users from interacting with the rest of the application until they dismiss the dialog

```
public class GUIDemo
{
    public static void main(String[] args)
    {
        String firstNumber = JOptionPane.showInputDialog("Enter 1st integer ");
        String secondNumber = JOptionPane.showInputDialog("Enter 2nd integer ");

        int number1 = Integer.parseInt(firstNumber);
        int number2 = Integer.parseInt(secondNumber);

        int sum = number1 + number2;

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





GUI - Dialog

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

`showMessageDialog` is a `static` method of `JOptionPane`.

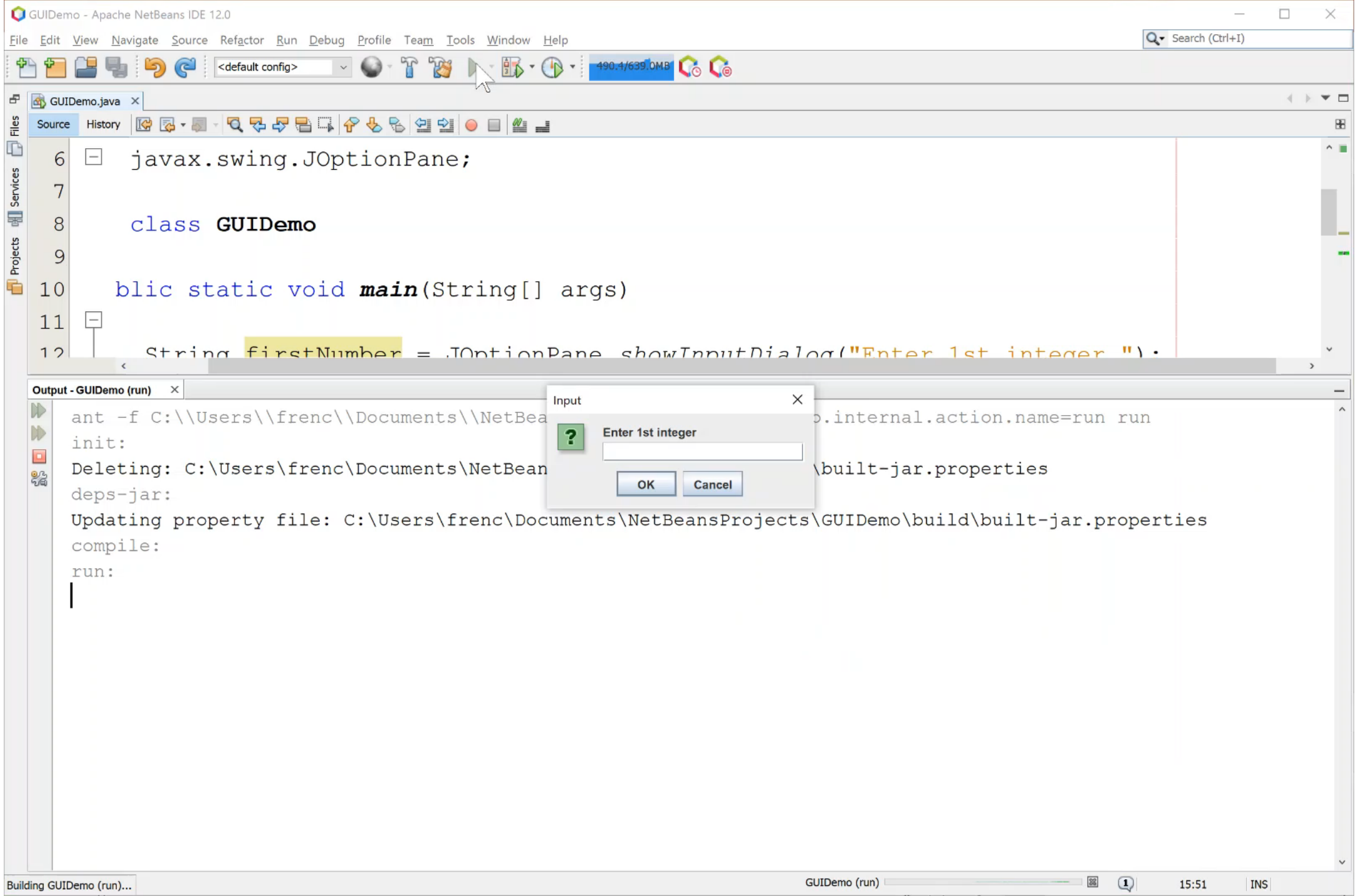
First argument

where to position the dialog box

a dialog is typically displayed from a GUI application within its own window

the first argument refers to that window (known as the parent window) and causes the dialog to appear centered over the parent

`null` causes the dialog to be displayed at the center of your screen


GUIDemo - Apache NetBeans IDE 12.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
490.4/639.0MB
GUIDemo.java
Source History
6 javax.swing.JOptionPane;
8 class GUIDemo
10 blic static void main(String[] args)
Output - GUIDemo (run)
ant -f C:\\Users\\frenc\\Documents\\NetBea .internal.action.name=run run
init:
Deleting: C:\Users\frenc\Documents\NetBean \built-jar.properties
deps-jar:
Updating property file: C:\Users\frenc\Documents\NetBeansProjects\GUIDemo\build\built-jar.properties
compile:
run:
Input
Enter 1st integer
OK
Cancel
Building GUIDemo (run)...
GUIDemo (run)
15:51
INS

GUI - Dialog

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

`showMessageDialog` is a `static` method of `JOptionPane`.

Second argument

message to display

`"The sum is " + sum`

Note where the comma is…

GUI - Dialog

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

`showMessageDialog` is a `static` method of `JOptionPane`.

Third argument

`String` that will appear in the title bar at the top of the dialog

GUI - Dialog

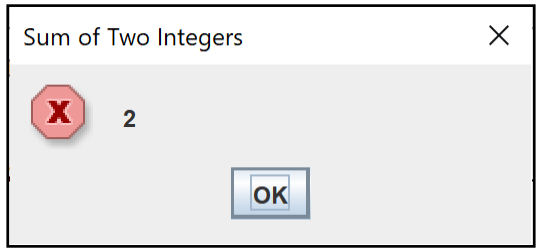
```
JOptionPane.showMessageDialog(null, "The sum is " + sum,
                "Sum of Two Integers", JOptionPane.PLAIN_MESSAGE);
```

`showMessageDialog` is a `static` method of `JOptionPane`.

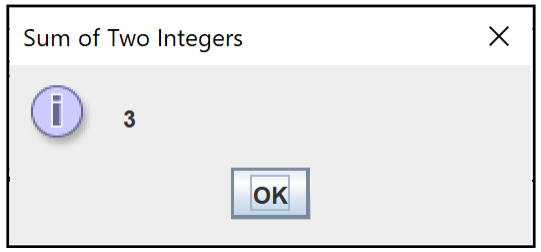
Fourth argument

type of message dialog box to display

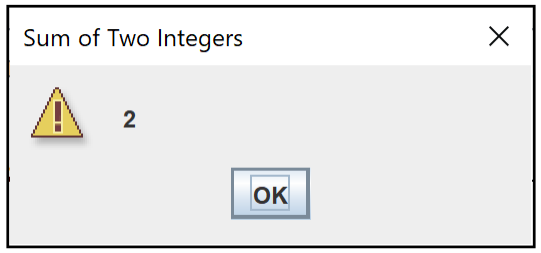
`PLAIN_MESSAGE` does not display an icon to the left of the message



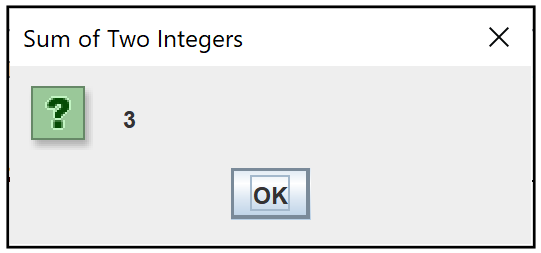
ERROR_MESSAGE



INFORMATION_MESSAGE



WARNING_MESSAGE



QUESTION_MESSAGE

Default icon for input dialog boxes

```
public class GUIDemo
{
    public static void main(String[] args)
    {
        String firstNumber = JOptionPane.showInputDialog("Enter 1st integer ");
        String secondNumber = JOptionPane.showInputDialog("Enter 2nd integer ");

        int number1 = Integer.parseInt(firstNumber);
        int number2 = Integer.parseInt(secondNumber);

        int sum = number1 + number2;

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

```
package guidemo;

import javax.swing.JOptionPane;

public class GUIDemo
{
    public static void main(String[] args)
    {
        int number1 = Integer.parseInt(JOptionPane.showInputDialog("Enter 1st integer "));
        int number2 = Integer.parseInt(JOptionPane.showInputDialog("Enter 2nd integer "));

        JOptionPane.showMessageDialog(null, number1+number2,
                "Sum of Two Integers", JOptionPane.PLAIN_MESSAGE);
    }
}
```

```
package guidemo;

import javax.swing.JOptionPane;

public class GUIDemo
{
    public static void main(String[] args)
    {
        JOptionPane.showMessageDialog(null,
            Integer.parseInt(JOptionPane.showInputDialog("Enter 1st integer "))+
            Integer.parseInt(JOptionPane.showInputDialog("Enter 2nd integer ")),
            "Sum of Two Integers", JOptionPane.PLAIN_MESSAGE);
    }
}
```

GUI

While it is possible to perform input and output using `JOptionPane` dialogs, we will want more elaborate user interfaces.

We are going to cover some basic Swing GUI components

Component	Description

GUI

Swing vs AWT

There are actually three sets of Java GUI components

Abstract Window Toolkit (AWT) in package `java.awt`

AWT existed before Swing and now there's JavaFX

GUI

AWT

AWT components look like the native GUI components of the platform on which a Java program executes.

A `Button` object displayed by a Java program running on Windows looks like other buttons in Windows apps

A `Button` object displayed by a Java program running on Mac looks like other buttons in Mac apps.

A component's appearance and the way in which the user interacts with it are known as its *look-and-feel*.

GUI

Swing

Swing GUI components allow you to specify a uniform *look-and-feel* for your application across all platforms or to use each platform's custom *look-and-feel*.

An application can even change the *look-and-feel* during execution to enable users to choose their preferred *look-and-feel*.

GUI

Lightweight vs Heavyweight GUI Components

Most Swing components are lightweight components

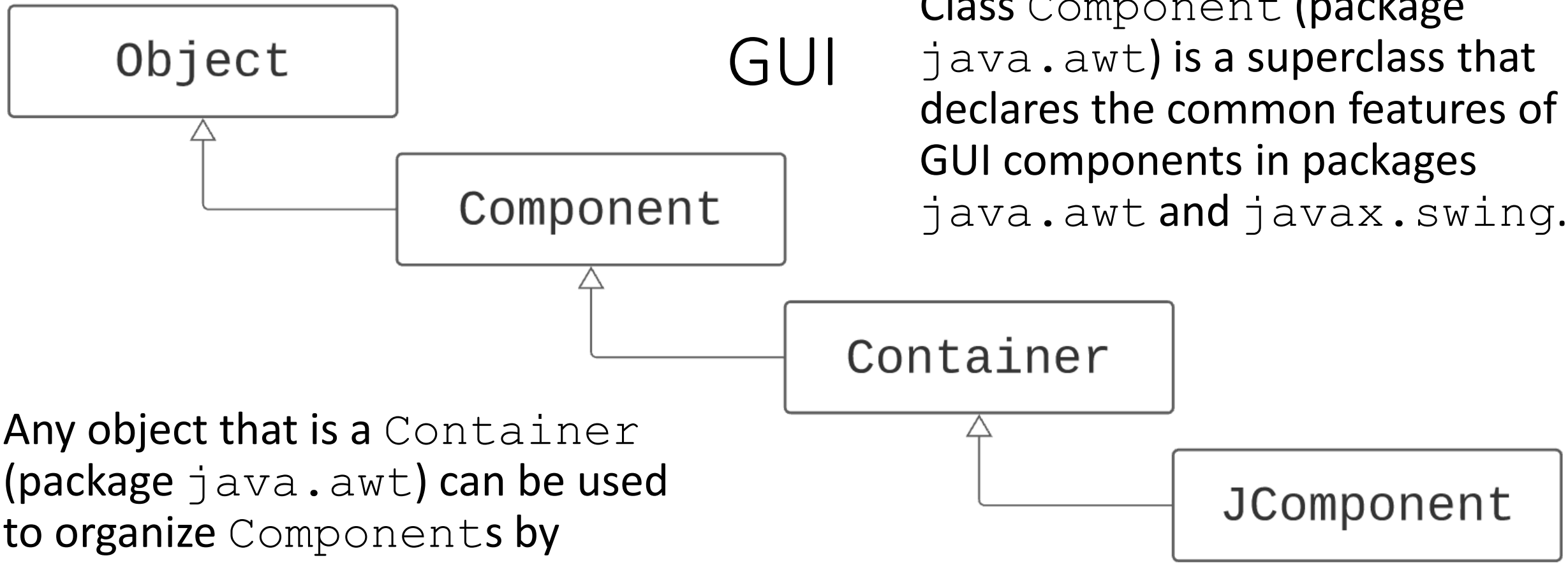
- they are written, manipulated and display completely in Java

AWT components are heavyweight components

- they rely on the local platform's windowing system to determine their functionality and their look-and-feel

Several Swing components are heavyweight components

GUI



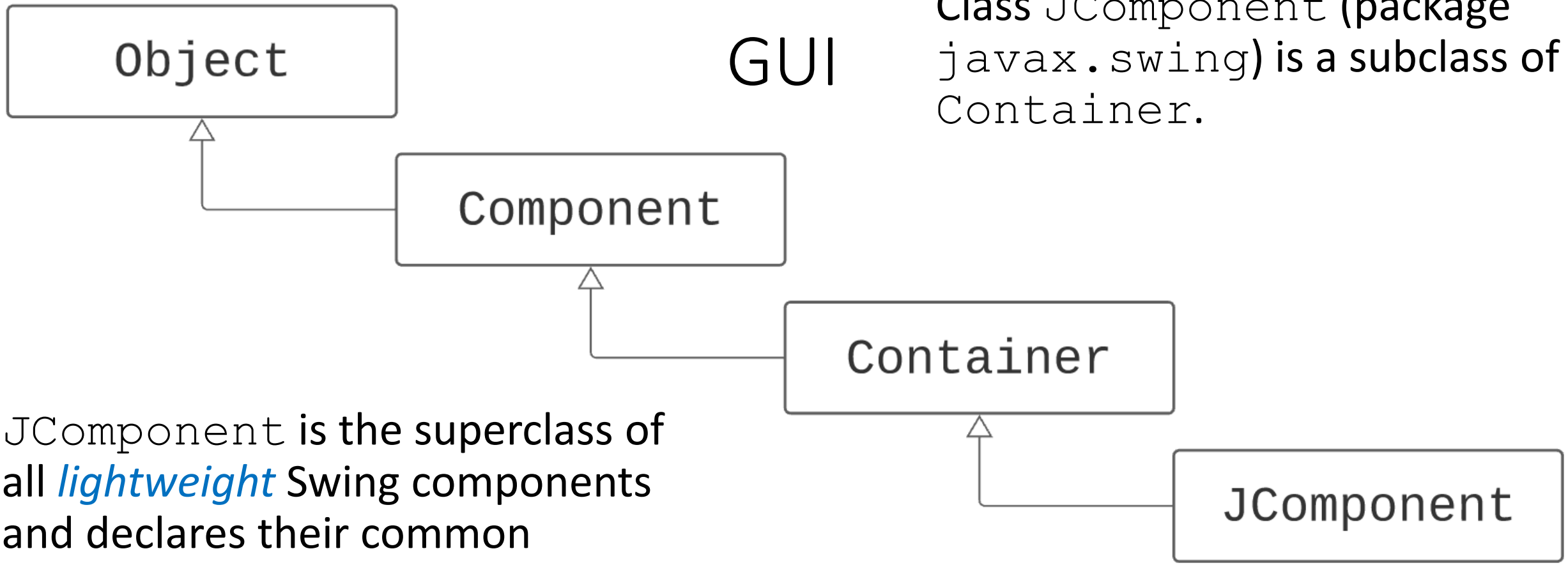
Class `Component` (package `java.awt`) is a superclass that declares the common features of GUI components in packages `java.awt` and `javax.swing`.

Any object that is a `Container` (package `java.awt`) can be used to organize `Component`s by attaching the `Component`s to the `Container`.

`Container`s can be placed in other `Container`s to organize a GUI

GUI

Class `JComponent` (package `javax.swing`) is a subclass of `Container`.



`JComponent` is the superclass of all *lightweight* Swing components and declares their common attributes and behaviors.

Because `JComponent` is a subclass of `Container`, all lightweight Swing components are also `Containers`.

GUI

Some common features supported by `JComponent` include:

1. A pluggable look-and-feel for customizing the appearance of components
2. Shortcut keys (called mnemonics) for direct access to GUI components through the keyboard
3. Brief descriptions of a GUI component's purpose (tool tips) that are displayed when the mouse cursor is positioned over the component for a short time
4. Support for accessibility such as braille screen readers for the visually impaired
5. Support for user-interface localization – customizing the user interface to display in different languages and use local cultural conventions

GUI

Displaying Text and Images in a Window

Most windows you create to contain Swing GUI components are instances of class `JFrame` or a subclass of `JFrame`

`JFrame` is an indirect subclass of class `java.awt.Window` that provides the basic attributes and behaviors of a window

a title bar at the top

buttons to minimize, maximize and close the window

GUI - Label

Displaying Text and Images in a Window

A typical GUI consists of many components.

These components are often labeled with text stating the purpose of the component.

This text is known as a *label* and is created using a `JLabel` which is a subclass of `JComponent`

A `JLabel` displays read-only text, an image or both

Applications rarely change a label's contents after creating it

GUI - Label

```
package labeltest;

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(460,380);
        labelFrame.setVisible(true);
    }
}
```

GUI - Label

```
LabelFrame labelFrame = new LabelFrame();
```

Instantiates an object named `labelFrame` from class `LabelFrame`

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

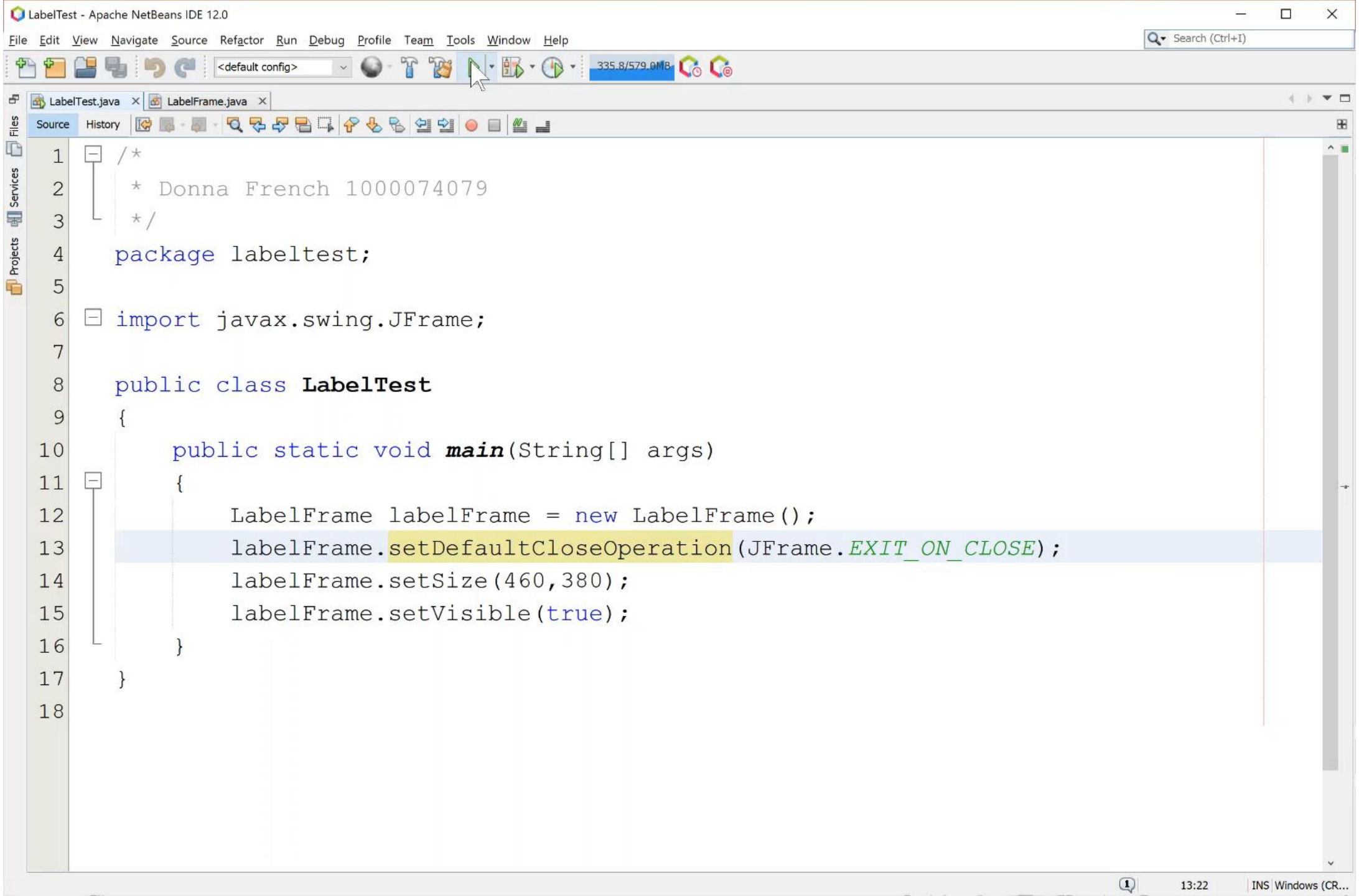
`EXIT_ON_CLOSE` is a constant that indicates that the program should terminate when the window is closed by the user. Without this line, the application will not terminate when the user closes the window.

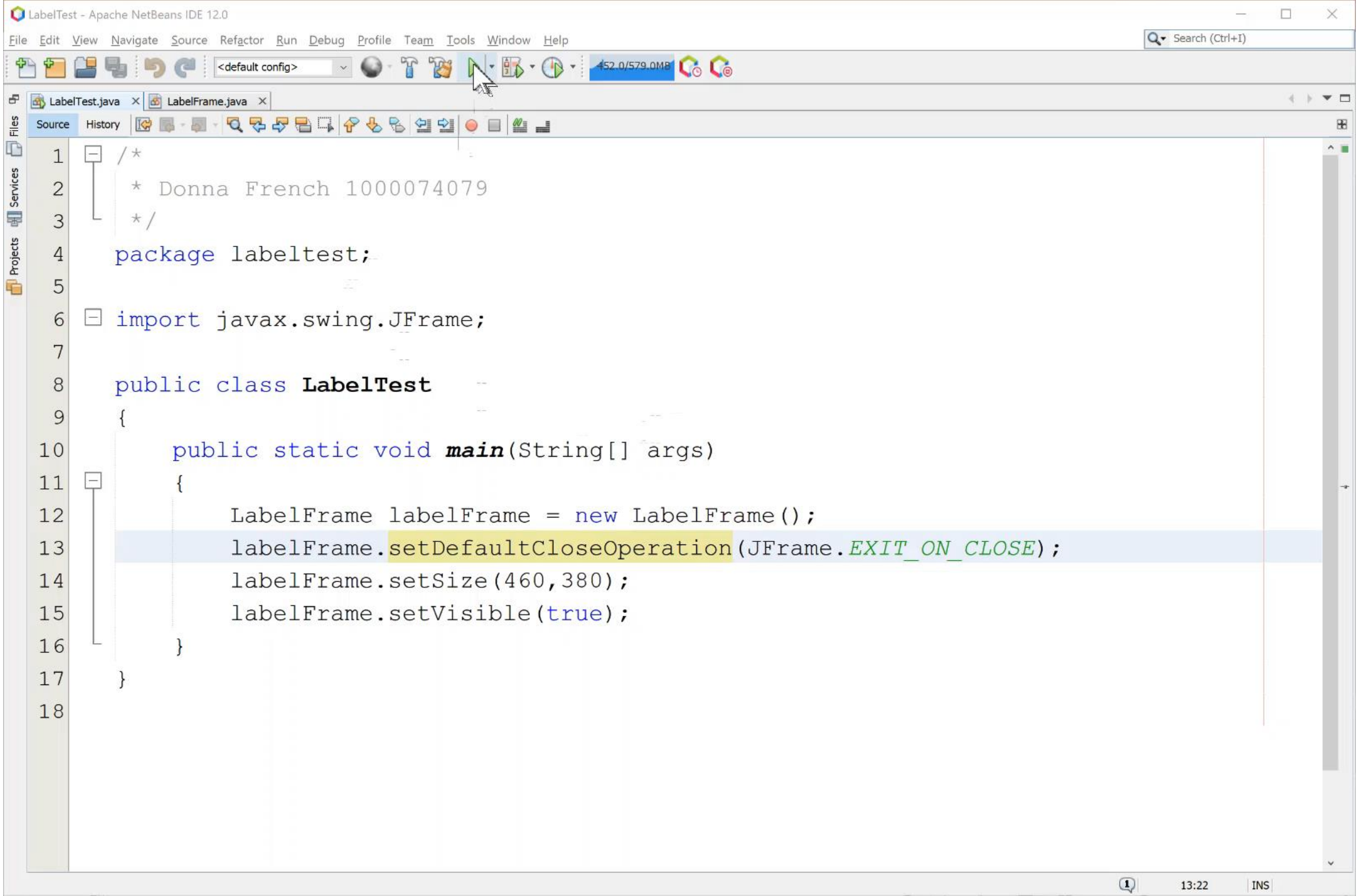
```
labelFrame.setSize(460,380);
```

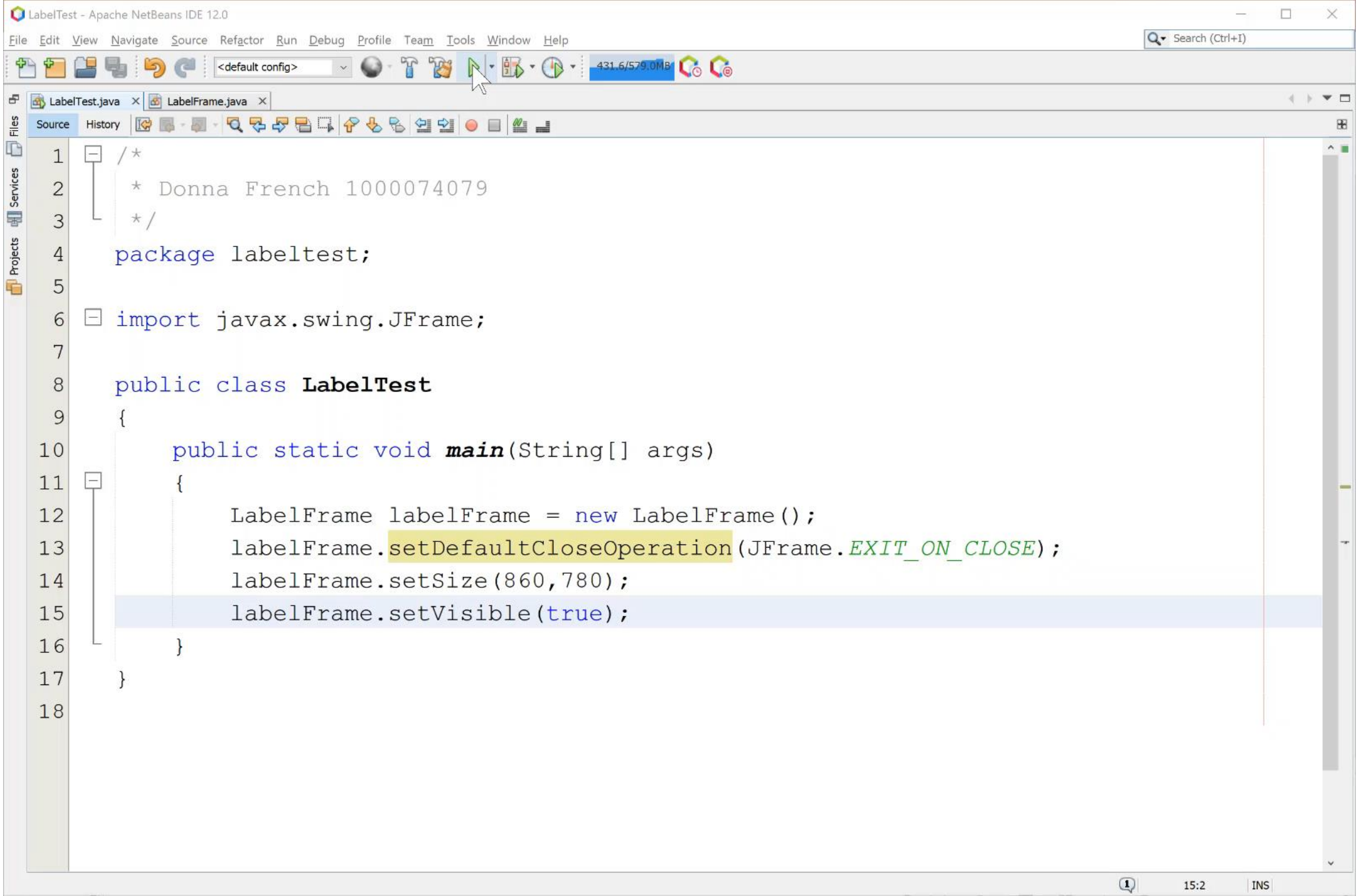
Specifies the width and height of the window in *pixels*

```
labelFrame.setVisible(true);
```

Sending `true` to `setVisible` causes the window to be displayed


LabelTest - Apache NetBeans IDE 12.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
335.8/579.0MB
LabelTest.java
LabelFrame.java
Source History
Files
Services
Projects
1 /*
2 * Donna French 1000074079
3 */
4 package labeltest;
5
6 import javax.swing.JFrame;
7
8 public class LabelTest
9 {
10 public static void main(String[] args)
11 {
12 LabelFrame labelFrame = new LabelFrame();
13 labelFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14 labelFrame.setSize(460,380);
15 labelFrame.setVisible(true);
16 }
17 }
18
13:22
INS Windows (CR...


LabelTest - Apache NetBeans IDE 12.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
452.0/579.0MB
LabelTest.java
LabelFrame.java
Source History
Files
Services
Projects
1 /*
2 * Donna French 1000074079
3 */
4 package labeltest;
5
6 import javax.swing.JFrame;
7
8 public class LabelTest
9 {
10 public static void main(String[] args)
11 {
12 LabelFrame labelFrame = new LabelFrame();
13 labelFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14 labelFrame.setSize(460,380);
15 labelFrame.setVisible(true);
16 }
17 }
18
13:22
INS


LabelTest - Apache NetBeans IDE 12.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
431.6/579.0MB
LabelTest.java
LabelFrame.java
Source History
Files
Services
Projects
1 /*
2 * Donna French 1000074079
3 */
4 package labeltest;
5
6 import javax.swing.JFrame;
7
8 public class LabelTest
9 {
10 public static void main(String[] args)
11 {
12 LabelFrame labelFrame = new LabelFrame();
13 labelFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14 labelFrame.setSize(860,780);
15 labelFrame.setVisible(true);
16 }
17 }
18
15:2
INS

GUI – Label

`LabelFrame` class

`import java.awt.FlowLayout;`	specifies how components are arranged
`import javax.swing.Icon;`	interface used to manipulate images
`import javax.swing.ImageIcon;`	loads image
`import javax.swing.JFrame;`	provides basic window features
`import javax.swing.JLabel;`	displays text and images
`import javax.swing.SwingConstants;`	common constants used with Swing

GUI – Label

`LabelFrame` class

```
public class LabelFrame extends JFrame
{
    private final JLabel label1;
    private final JLabel label2;
    private final JLabel label3;
```

`LabelFrame` inherits from `JFrame`

Once these variables are set to a value, they cannot change*.

*Label text typically does not change

```
public LabelFrame()
{
    super("Testing JLabel");
    setLayout(new FlowLayout());

    label1 = new JLabel("Label with text");
    label1.setToolTipText("This is label1");
    add(label1);

    Icon BB = new ImageIcon(getClass().getResource("BinaryBlaze.png"));
    label2 = new JLabel("Label with text and icon", BB, SwingConstants.LEFT);
    label2.setToolTipText("This is label2");
    add(label2);

    label3 = new JLabel();
    label3.setText("Label with icon and text at bottom");
    label3.setIcon(BB);
    label3.setHorizontalTextPosition(SwingConstants.CENTER);
    label3.setVerticalTextPosition(SwingConstants.BOTTOM);
    label3.setToolTipText("This is label3");
    add(label3);
}
```

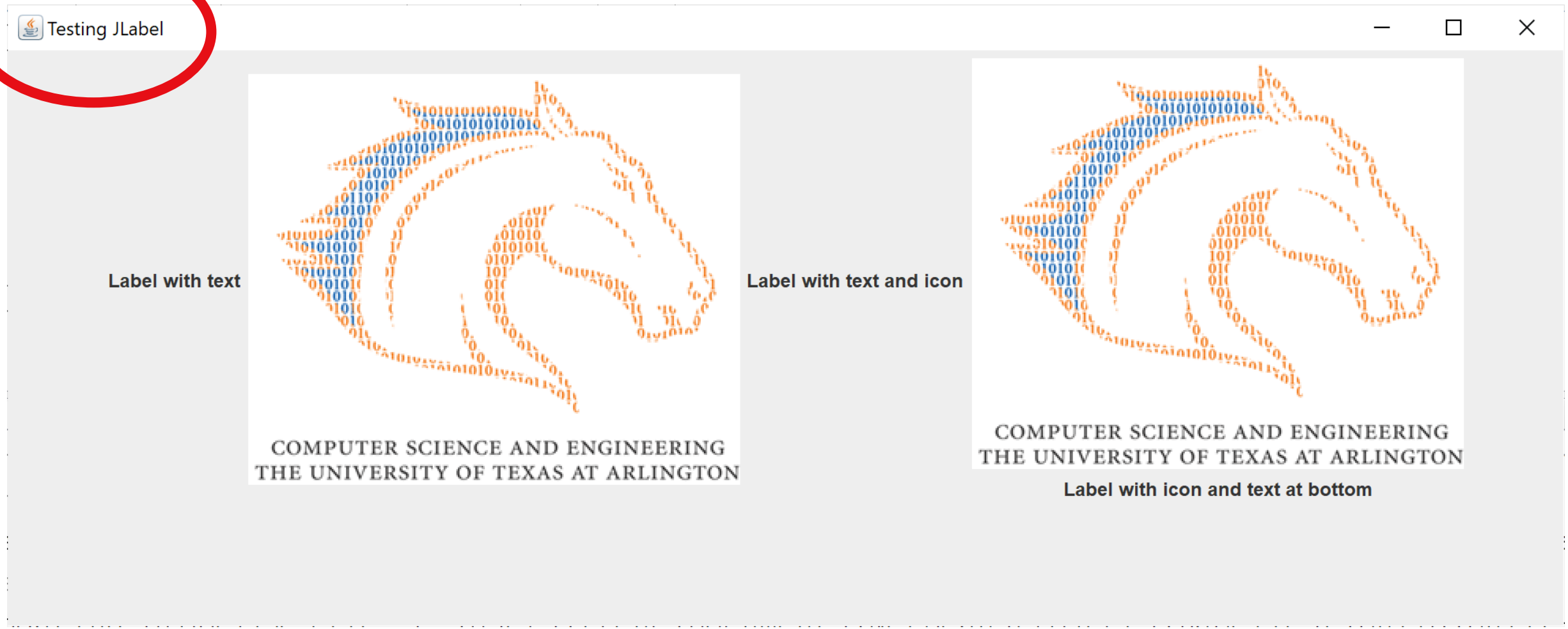
GUI - Label

```
public LabelFrame()
{
    super("Testing JLabel");
    setLayout(new FlowLayout());
```

Invoke superclass `JFrame`'s constructor with the argument `"Testing JLabel"`

`JFrame`'s constructor uses this `String` as the text in the window's title bar.

```
super("Testing JLabel");
```



GUI - Label

```
public LabelFrame()
{
    super("Testing JLabel");
    setLayout(new FlowLayout());
```

When building a GUI, you must attach each GUI component to a container.

You also typically decide where to position each GUI component

This is known as *specifying the layout*

Java provides several layout managers

GUI - Label

```
public LabelFrame()
{
    super("Testing JLabel");
    setLayout(new FlowLayout());
```

`FlowLayout` is a layout manager

components are placed in a container from left to right in the order in which they are added

when no more components can fit on the current line, they continue to be displayed left to right on the next line

GUI - Label

```
public LabelFrame()
{
    super("Testing JLabel");
    setLayout(new FlowLayout());
```

`FlowLayout` is a layout manager

- if the container is resized, `FlowLayout` *reflows* the components
- method `setLayout` is inherited from class `Container`
- argument must be an object of a class that implements the `LayOutManager` interface (in our example, that's `FlowLayout`)

GUI - Label

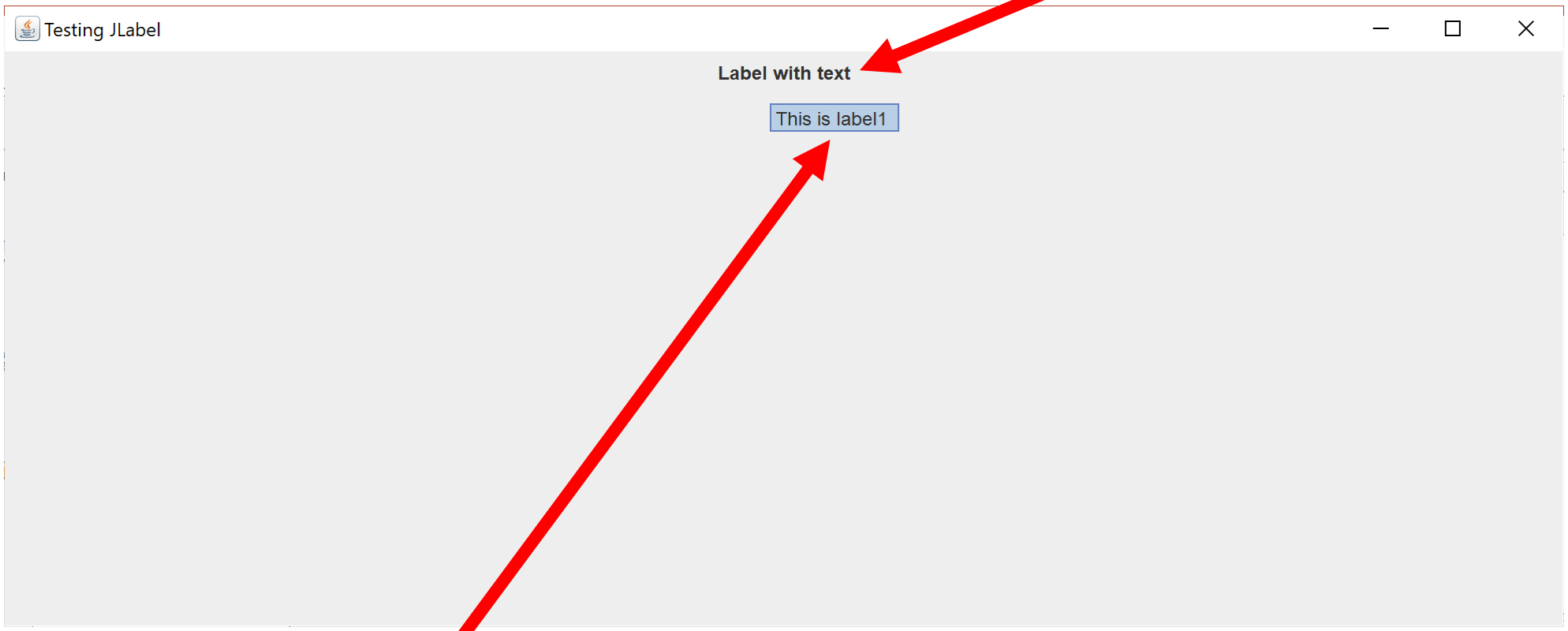
```
label1 = new JLabel("Label with text");
label1.setToolTipText("This is label1");
add(label1);
```

Create a new `JLabel` and pass "`Label with text`" to the constructor

`setToolTipText()` sets the tool tip that displays when the user positions the mouse cursor over the `JLabel` in the GUI

`add()` attaches `label1` to the `LabelFrame`

Text sent to `JLabel`'s constructor



Tool tip given to `setToolTipText()`

GUI - Label

```
Icon BB = new ImageIcon(getClass().getResource("BinaryBlaze.png"));
label2 = new JLabel("Label with text and icon", BB, SwingConstants.LEFT);
label2.setToolTipText("This is label2");
add(label2);
```

`getClass()` gets a reference to the class which `getResource()` then uses to look for the file in the same folder/directory as the class itself.

`BB` is an object of class `ImageIcon` which is a class that can hold an image.

Create a new `JLabel` by passing "`Label with text and icon`", the `ImageIcon` object `BB` and the `SwingConstant LEFT` to the constructor

`add()` attaches `label2` to the `LabelFrame`

javax.swing

Class JLabel

java.lang.Object
  java.awt.Component
    java.awt.Container
      javax.swing.JComponent
        javax.swing.JLabel

All Implemented Interfaces:

ImageObserver, MenuContainer, Serializable, Accessible, SwingConstants

Direct Known Subclasses:

BasicComboBoxRenderer, DefaultListCellRenderer, DefaultTableCellRenderer, DefaultTreeCellRenderer

---

```
public class JLabel
extends JComponent
implements SwingConstants, Accessible
```

A display area for a short text string or an image, or both. A label does not react to input events. As a result, it cannot get the keyboard focus. A label can, however, display a keyboard alternative as a convenience for a nearby component that has a keyboard alternative but can't display it.

A `JLabel` object can display either text, an image, or both. You can specify where in the label's display area the label's contents are aligned by setting the vertical and horizontal alignment. By default, labels are vertically centered in their display area. Text-only labels are leading edge aligned, by default; image-only labels are horizontally centered, by default.

You can also specify the position of the text relative to the image. By default, text is on the trailing edge of the image, with the text and image vertically aligned.

Constructor Summary

Constructors

Constructor and Description
`JLabel()` Creates a `JLabel` instance with no image and with an empty string for the title.
`JLabel(Icon image)` Creates a `JLabel` instance with the specified image.
`JLabel(Icon image, int horizontalAlignment)` Creates a `JLabel` instance with the specified image and horizontal alignment.
`JLabel(String text)` Creates a `JLabel` instance with the specified text.
`JLabel(String text, Icon icon, int horizontalAlignment)` Creates a `JLabel` instance with the specified text, image, and horizontal alignment.
`JLabel(String text, int horizontalAlignment)` Creates a `JLabel` instance with the specified text and horizontal alignment.

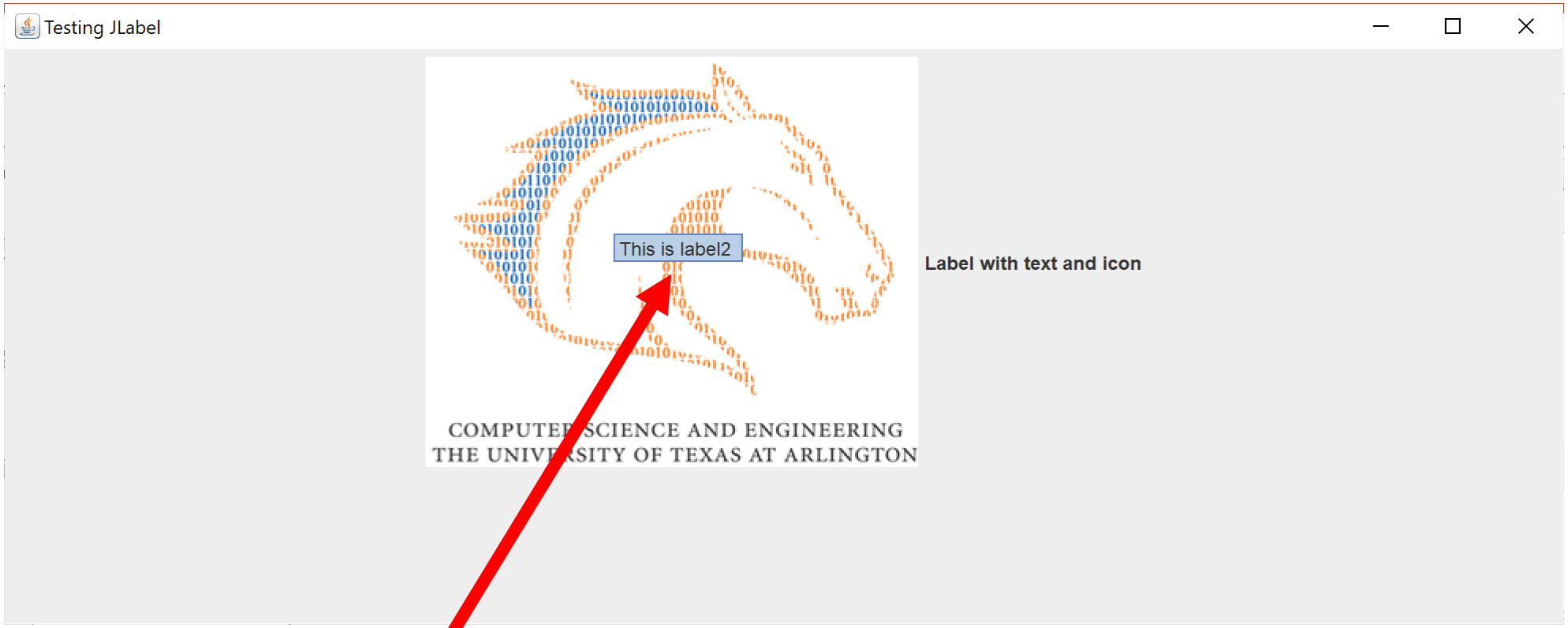
SwingConstants

Field Summary

Fields

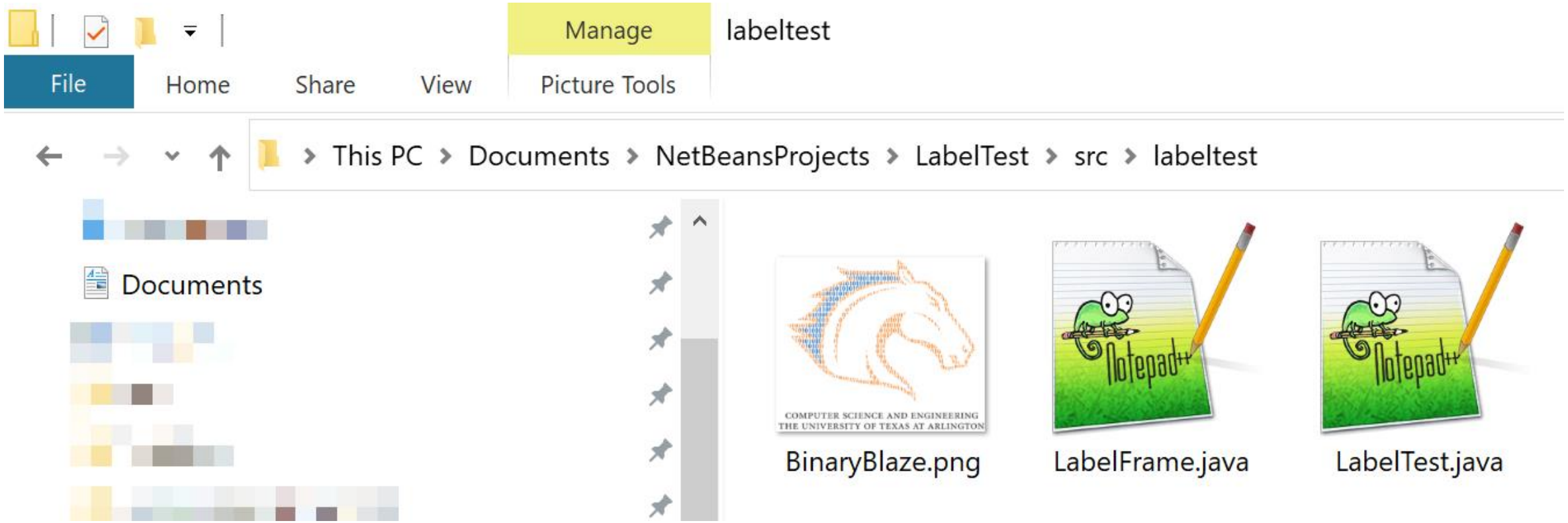
Modifier and Type	Field and Description
static int	**BOTTOM** Box-orientation constant used to specify the bottom of a box.
static int	**CENTER** The central position in an area.
static int	**EAST** Compass-direction east (right).
static int	**HORIZONTAL** Horizontal orientation.
static int	**LEADING** Identifies the leading edge of text for use with left-to-right and right-to-left languages.
static int	**LEFT** Box-orientation constant used to specify the left side of a box.
static int	**NEXT** Identifies the next direction in a sequence.
static int	**NORTH** Compass-direction North (up).
static int	**NORTH_EAST** Compass-direction north-east (upper right).
static int	**NORTH_WEST** Compass-direction north west (upper left).
static int	**PREVIOUS** Identifies the previous direction in a sequence.
static int	**RIGHT** Box-orientation constant used to specify the right side of a box.
static int	**SOUTH** Compass-direction south (down).
static int	**SOUTH_EAST** Compass-direction south-east (lower right).
static int	**SOUTH_WEST** Compass-direction south-west (lower left).
static int	**TOP** Box-orientation constant used to specify the top of a box.
static int	**TRAILING** Identifies the trailing edge of text for use with left-to-right and right-to-left languages.
static int	**VERTICAL** Vertical orientation.
static int	**WEST** Compass-direction west (left).

```
Icon BB = new ImageIcon(getClass().getResource("BinaryBlaze.png"));
label2 = new JLabel("Label with text and icon", BB, SwingConstants.LEFT);
label2.setToolTipText("This is label2");
add(label2);
```

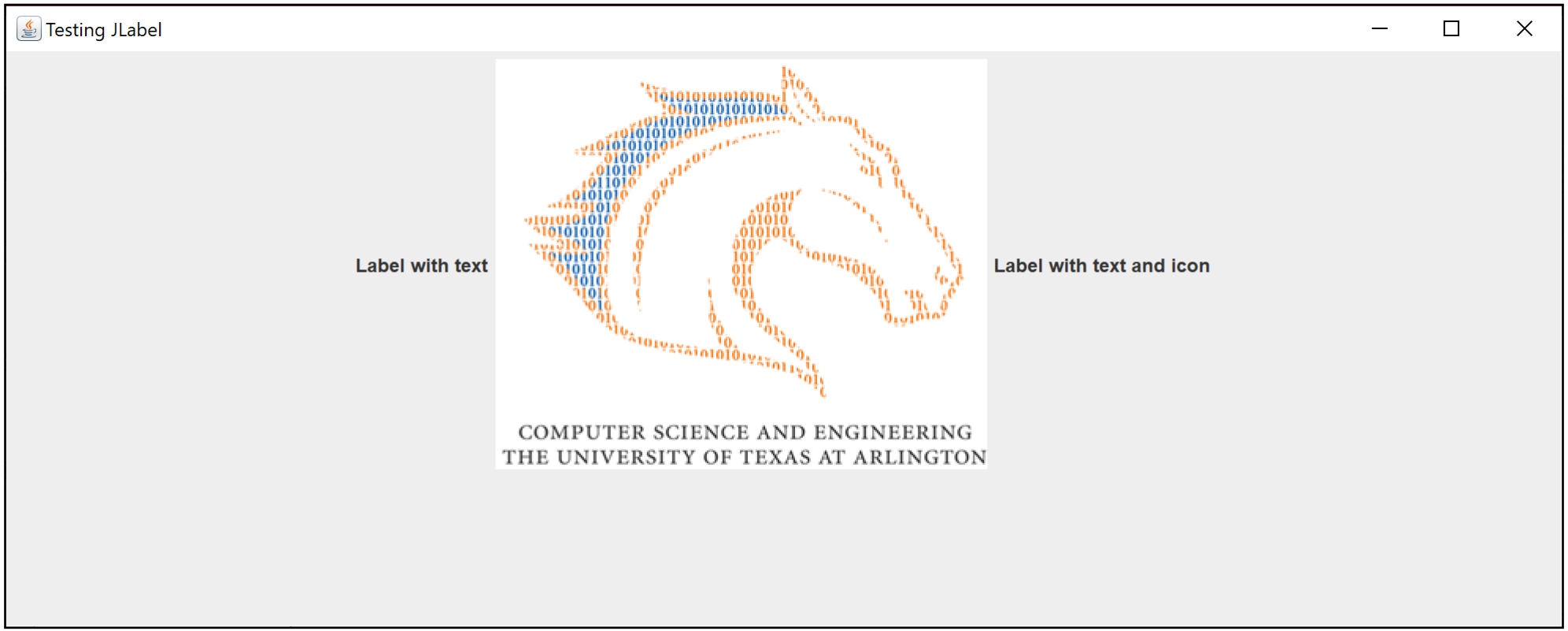


Tool tip given to `setToolTipText()`

```
Icon BB = new ImageIcon(getClass().getResource("BinaryBlaze.png"));
label2 = new JLabel("Label with text and icon", BB, SwingConstants.LEFT);
label2.setToolTipText("This is label2");
add(label2);
```



```
label1 = new JLabel("Label with text");
label1.setToolTipText("This is label1");
add(label1);

Icon BB = new ImageIcon(getClass().getResource("BinaryBlaze.png"));
label2 = new JLabel("Label with text and icon", BB, SwingConstants.LEFT);
label2.setToolTipText("This is label2");
add(label2);
```

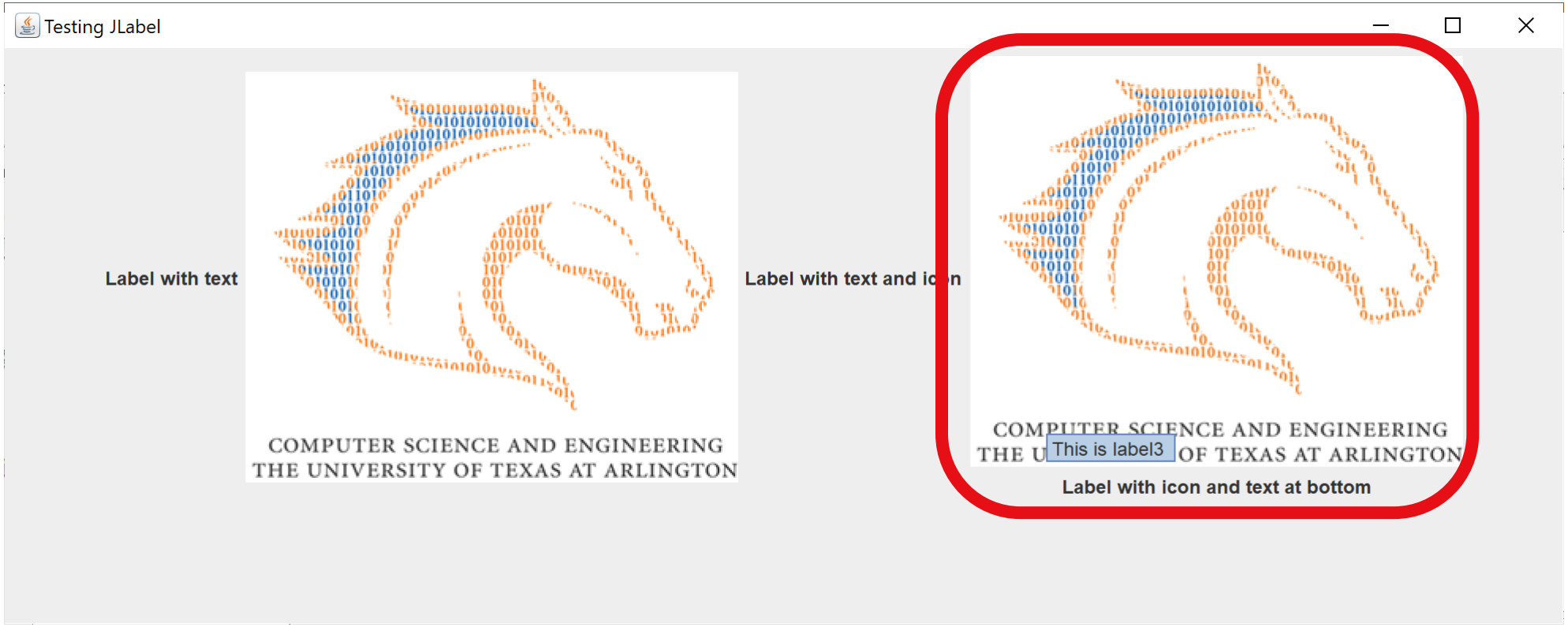


GUI - Label

```
label3 = new JLabel();
label3.setText("Label with icon and text at bottom");
label3.setIcon(BB);
label3.setHorizontalTextPosition(SwingConstants.CENTER);
label3.setVerticalTextPosition(SwingConstants.BOTTOM);
label3.setToolTipText("This is label3");
add(label3);
```

`label3` shows how to construct a basic `JLabel` and then use methods to set various aspects of it after construction.

```
label3 = new JLabel();
label3.setText("Label with icon and text at bottom");
label3.setIcon(BB);
label3.setHorizontalTextPosition(SwingConstants.CENTER);
label3.setVerticalTextPosition(SwingConstants.BOTTOM);
label3.setToolTipText("This is label3");
add(label3);
```

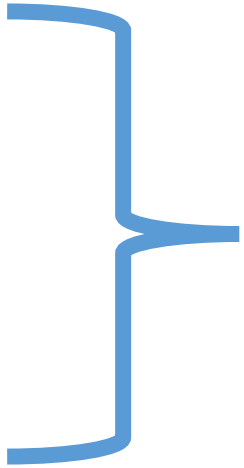


Event Handling

GUIs are event driven

user does something to cause the GUI to react and do something

clicking a button
typing in a text field
select an item from a menu
closing a window
moving the mouse

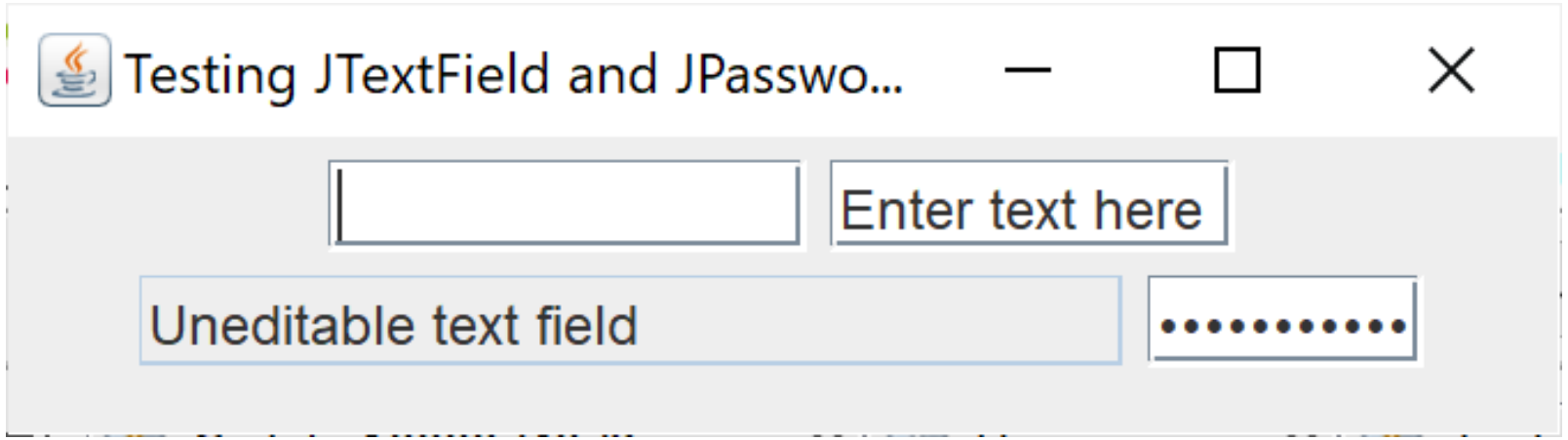


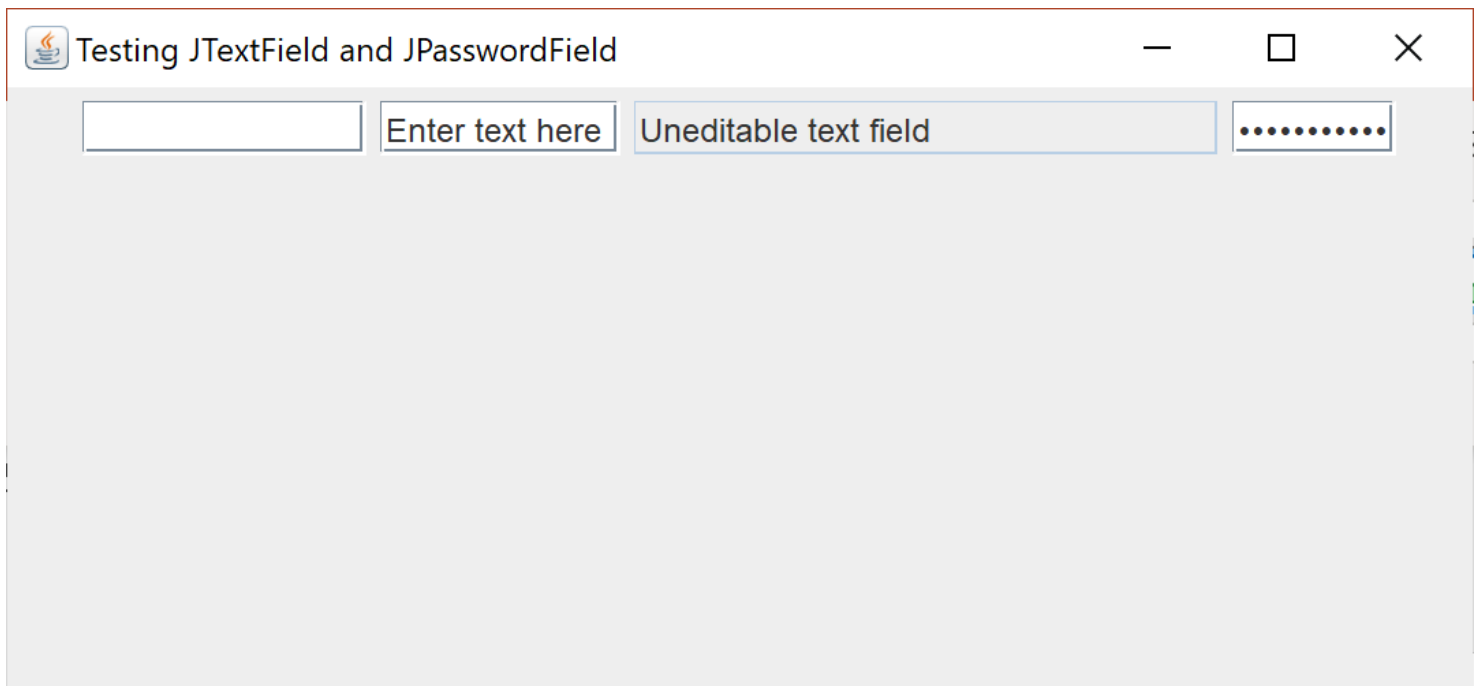
events

The code that performs a task in response to an event is called an ***event handler*** and the process of responding to events is known as ***event handling***.

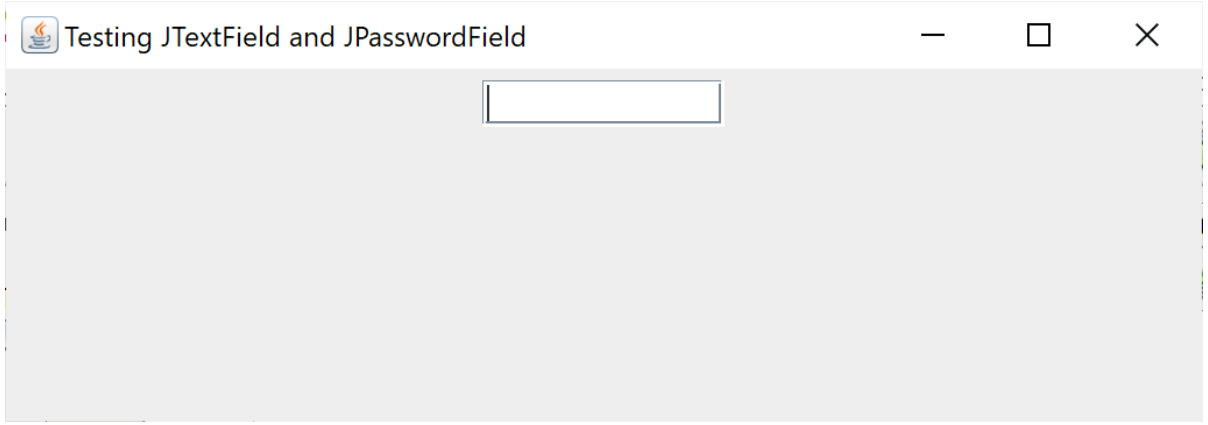
```
public class TextFieldFrame extends JFrame
{
    private final JTextField textField1;
    private final JTextField textField2;
    private final JTextField textField3;
    private final JPasswordField passwordField;
```

```
public TextFieldFrame()
{
    super("Testing JTextField and JPasswordField");
    setLayout(new FlowLayout());

    textField1 = new JTextField(10);
    add(textField1);

    textField2 = new JTextField("Enter text here ");
    add(textField2);

    textField3 = new JTextField("Uneditable text field", 21);
    textField3.setEditable(false);
    add(textField3);

    passwordField = new JPasswordField("Hidden text");
    add(passwordField);

    TextFieldHandler handler = new TextFieldHandler();
    textField1.addActionListener(handler);
    textField2.addActionListener(handler);
    textField3.addActionListener(handler);
    passwordField.addActionListener(handler);
}
```

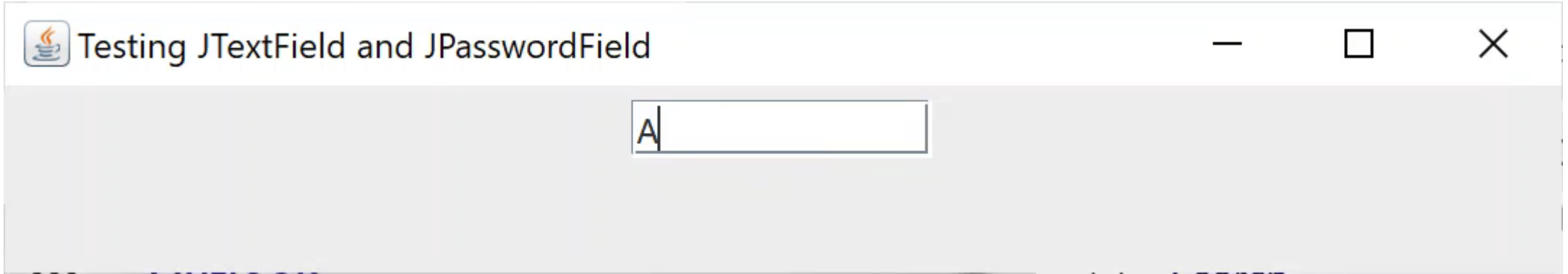

Testing JTextField and JPasswo...
Enter text here
Uneditable text field


Testing JTextField and JPasswordField
Enter text here
Uneditable text field

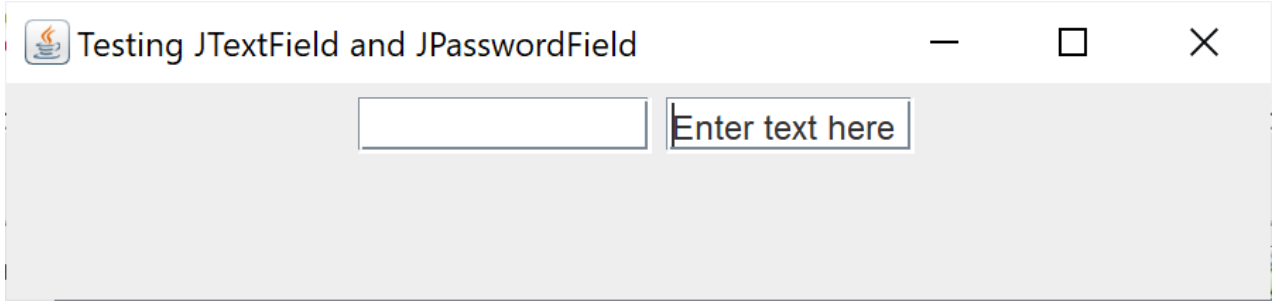
```
public TextFieldFrame()
{
    super("Testing JTextField and JPasswordField");
    setLayout(new FlowLayout());

    textField1 = new JTextField(10);
    add(textField1);
```



10 is the text column's width in pixels as determined by the average width of a character in the text's field's current font



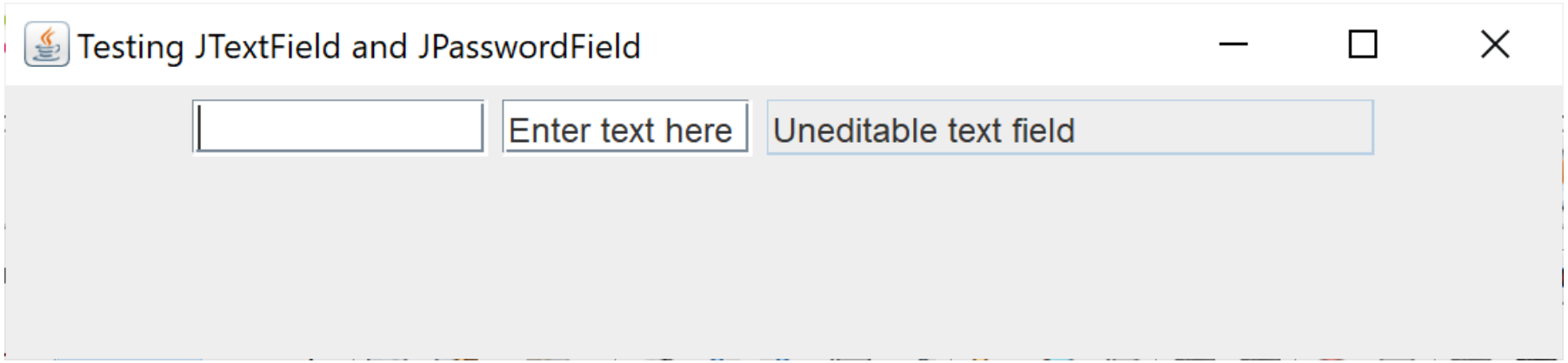
```
public TextFieldFrame()
{
    super("Testing JTextField and JPasswordField");
    setLayout(new FlowLayout());

    textField1 = new JTextField(10);
    add(textField1);

    textField2 = new JTextField("Enter text here ");
    add(textField2);
```



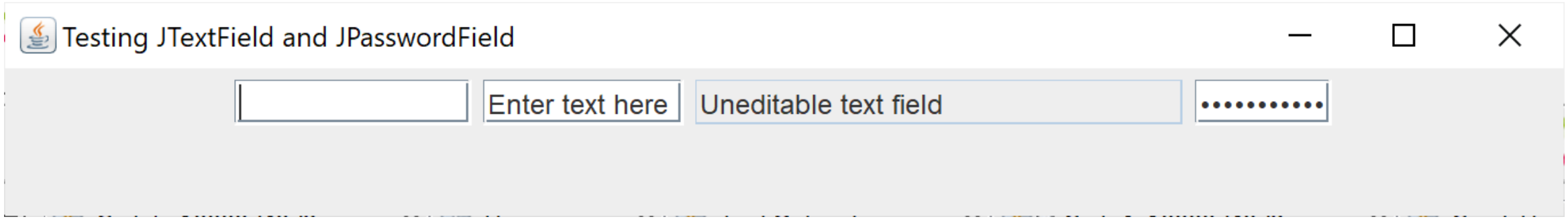
Text box is as wide as the text string given to the constructor.

```
public TextFieldFrame()
{
    super("Testing JTextField and JPasswordField");
    setLayout(new FlowLayout());

    textField1 = new JTextField(10);
    add(textField1);

    textField2 = new JTextField("Enter text here ");
    add(textField2);

    textField3 = new JTextField("Uneditable text field", 21);
    textField3.setEditable(false);
    add(textField3);
```

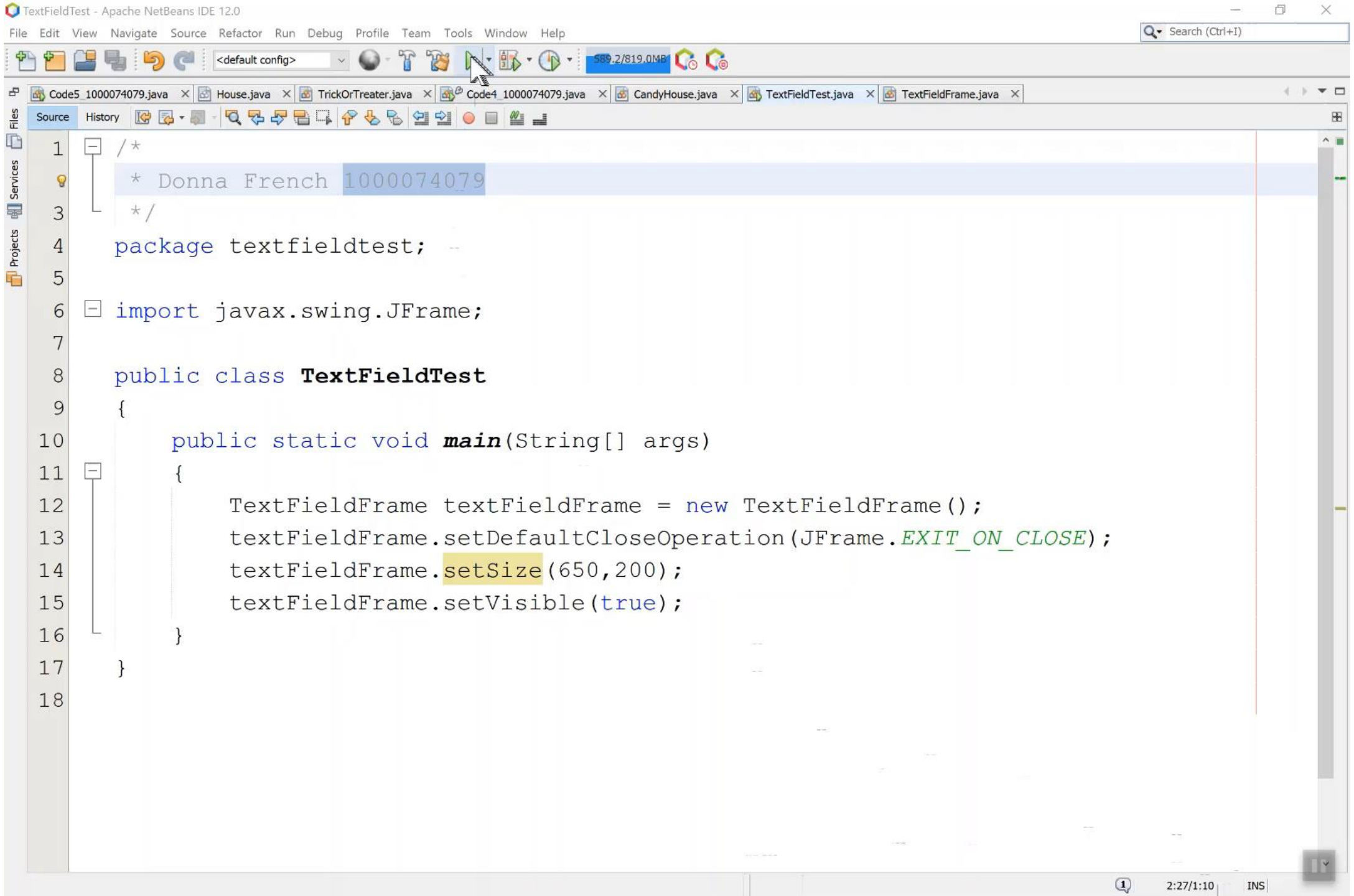
Field has a width of 21 but is set to uneditable.



```
passwordField = new JPasswordField("Hidden text");
add(passwordField);
```



```
TextFieldHandler handler = new TextFieldHandler();
textField1.addActionListener(handler);
textField2.addActionListener(handler);
textField3.addActionListener(handler);
passwordField.addActionListener(handler);
```


TextFieldTest - Apache NetBeans IDE 12.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
589.2/819.0MB
Code5_1000074079.java
House.java
TrickOrTreater.java
Code4_1000074079.java
CandyHouse.java
TextFieldTest.java
TextFieldFrame.java
Source History
Files
Services
Projects
1 /*
2 * Donna French 1000074079
3 */
4 package textfieldtest;
5
6 import javax.swing.JFrame;
7
8 public class TextFieldTest
9 {
10 public static void main(String[] args)
11 {
12 TextFieldFrame textFieldFrame = new TextFieldFrame();
13 textFieldFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14 textFieldFrame.setSize(650,200);
15 textFieldFrame.setVisible(true);
16 }
17 }
18
2:27/1:10
INS

Steps Required to Set Up Event Handling for a GUI Component

1. Create a class that represents the event handler and implements an appropriate interface – known as an ***event-listener interface***.

2. Indicate that an object of the class from Step 1 should be notified when the event occurs – known as ***registering the event handler***.

Nested Classes

All the classes we have discussed so far have been top-level classes.

meaning they were not declared inside another class

Java allows us to declare classes *inside* other classes

classes inside other classes are called ***nested classes***

Nested classes can be `static` or non-`static`

Non-`static` nested classes are called ***inner classes*** and are frequently used to implement event handlers.

Nested Classes

An inner class object must be created by an object of the top level class that contains the inner class.

Each inner class object implicitly has a reference to an object of its top level class.

The inner class is allowed to use this implicit reference to directly access all the variables and methods of the top level class.

A nested class that's `static` does not require an object of its top level class and does not implicitly have a reference to an object of the top level class.

TextFieldTest - Apache NetBeans IDE 12.0

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

<default config> 582.3/819.0MB

Code5_1000074079.java | House.java | TrickOrTreater.java | Code4_1000074079.java | CandyHouse.java | TextFieldTest.java | TextFieldFrame.java

Source History

```
5
6   import java.awt.FlowLayout;
7   import java.awt.event.ActionEvent;
8   import java.awt.event.ActionListener;
9   import javax.swing.JFrame;
10  import javax.swing.JOptionPane;
11  import javax.swing.JPasswordField;
12  import javax.swing.JTextField;
13
14  public class TextFieldFrame extends JFrame
15  {
16      private final JTextField textField1;
17      private final JTextField textField2;
18      private final JTextField textField3;
19      private final JPasswordField passwordField;
20
21      public TextFieldFrame()
22      {
23          super("Testing JTextField and JPasswordField");
24          setLayout(new FlowLayout());
25
26          textField1 = new JTextField(10);
```

25:1 INS Windows (CR...

Inner Class `TextFieldHandler`

The event handling is performed by an object of the private inner class `TextFieldHandler`.

This class is `private` because it will only be used to create event handlers for the text fields in the top level class `TextFieldFrame`

Event handling tends to be specific to the application in which they are defined so they are usually implemented as `private` inner classes or as anonymous classes.

Inner Class `TextFieldHandler`

GUI components can generate many events in response to user interactions

Each event is represented by a class and can be processed only by the appropriate type of event handler

When the user presses `ENTER` in a `JTextField` or `JPasswordField`, an `ActionEvent` occurs

An `ActionEvent` is processed by an object that implements the interface `ActionListener`

Inner Class `TextFieldHandler`

To handle the events, inner class `TextFieldHandler` implements interface `ActionListener` and overrides the only method in that interface – `actionPerformed`

`actionPerformed` specifies the tasks to perform when an `ActionEvent` occurs

Inner class `TextFieldHandler` fulfills Step 1 of setting up event handling…

> Create a class that represents the event handler and implements an appropriate interface – known as an ***event-listener interface***.

```
private class TextFieldHandler implements ActionListener
{
    @Override
    public void actionPerformed(ActionEvent event)
    {
        String string = "";

        if (event.getSource() == textField1)
            string = String.format("textField1: %s", event.getActionCommand());
        else if (event.getSource() == textField2)
            string = String.format("textField2: %s", event.getActionCommand());
        else if (event.getSource() == textField3)
            string = String.format("textField3: %s", event.getActionCommand());
        else if (event.getSource() == passwordField)
            string = String.format("passwordField: %s", event.getActionCommand());

        JOptionPane.showMessageDialog(null, string);
    }
}
```

This code is in `TextFieldFrame`'s constructor

```
TextFieldHandler handler = new TextFieldHandler();
textField1.addActionListener(handler);
textField2.addActionListener(handler);
textField3.addActionListener(handler);
passwordField.addActionListener(handler);
```

Registering the Event Handler for Each Text Field

```
TextFieldHandler handler = new TextFieldHandler();
```

instantiates a `TextFileHandler` object named `handler`

`handler`'s `actionPerformed` method wil be called automatically when the user presses `ENTER` in any of the GUI's text fields

Before this can occur, the program must register `handler` as the event handler for each text field

Registering the Event Handler for Each Text Field

```
textField1.addActionListener(handler);
textField2.addActionListener(handler);
textField3.addActionListener(handler);
passwordField.addActionListener(handler);
```

These event registration statements specify `handler` as the event handler for the three `JTextField`s and `JPasswordField`.

`JTextField`'s `addActionListener` is called to register the event handler for each component

Registering the Event Handler for Each Text Field

```
TextFieldHandler handler = new TextFieldHandler();
textField1.addActionListener(handler);
textField2.addActionListener(handler);
textField3.addActionListener(handler);
passwordField.addActionListener(handler);
```

`TextFieldHandler()` implements `ActionListener` so `handler` is an `ActionListener` object which is why you can pass it to `addActionListener`.

When `ENTER` is pressed, method `actionPerformed` in class `TextFieldHandler` is called to handle the event.

If an event handler is not registered for a text field, then the event is ignored.

```
private class TextFieldHandler implements ActionListener
{
   @Override
   public void actionPerformed(ActionEvent event)
   {
      String string = "";

      if (event.getSource() == textField1)
          string = String.format("textField1: %s", event.getActionCommand());
      else if (event.getSource() == textField2)
          string = String.format("textField2: %s", event.getActionCommand());
      else if (event.getSource() == textField3)
         string = String.format("textField3: %s", event.getActionCommand());
      else if (event.getSource() == passwordField)
         string = String.format("passwordField: %s", event.getActionCommand());

      JOptionPane.showMessageDialog(null, string);
   }
}
```

```
if (event.getSource() == textField1)
   string = String.format("textField1: %s", event.getActionCommand());
```

Method `getsource()` returns a reference to the event source

We can test if that reference matches one of our text field components

Method `getActionCommand()` is then used to obtain the text the user typed into the text field

We then use

```
JOptionPane.showMessageDialog(null, string);
```

to display the string that was created.

JButton

A button is a component the user clicks to trigger a specific action.

A Java application can use several types of buttons

- command buttons
- checkboxes
- toggle buttons
- radio buttons

A command button is typically used to initiate a command.

JButton

A command button generates an `ActionEvent` when the user clicks it

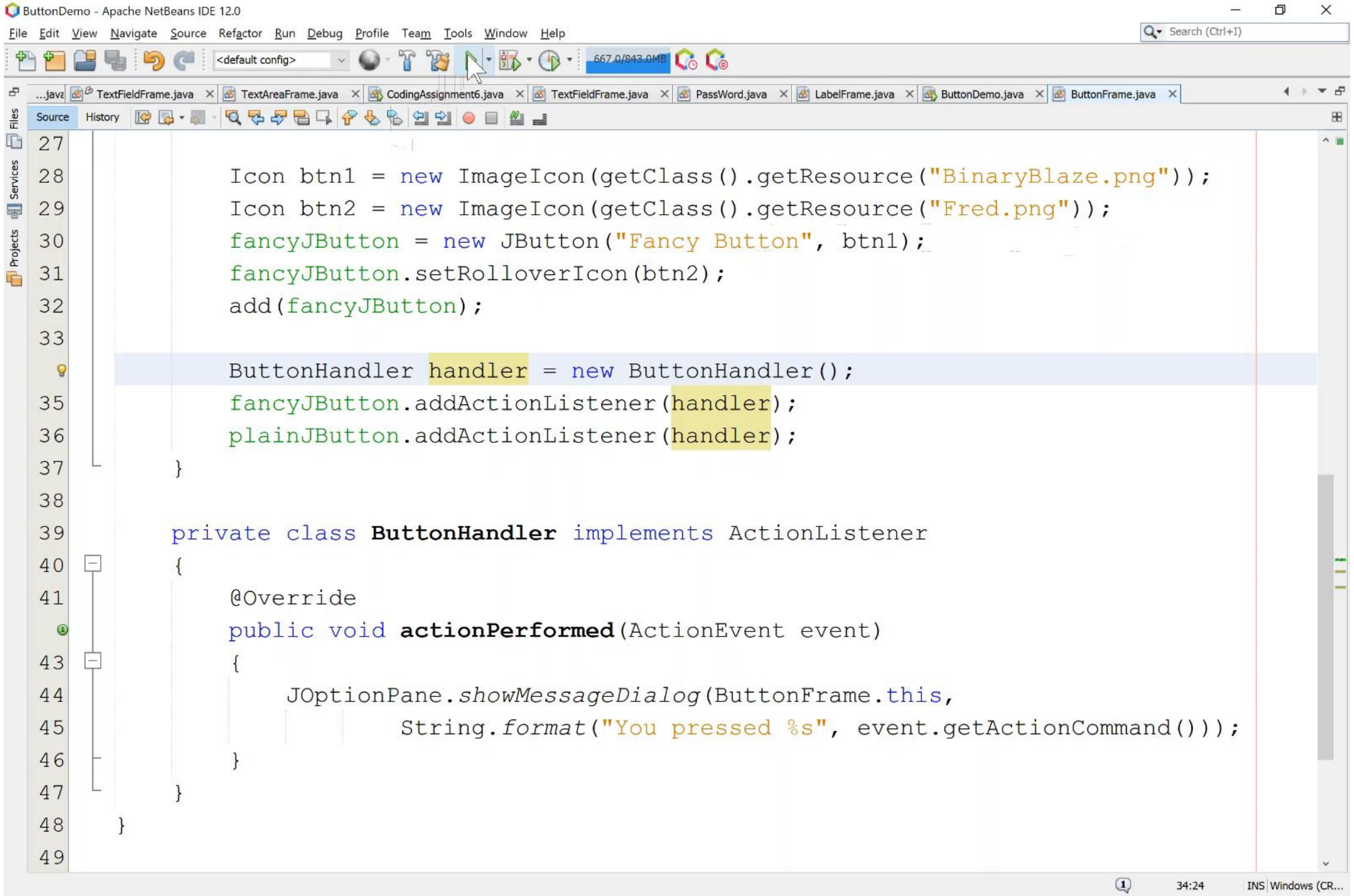
Command buttons are created with class `JButton`.

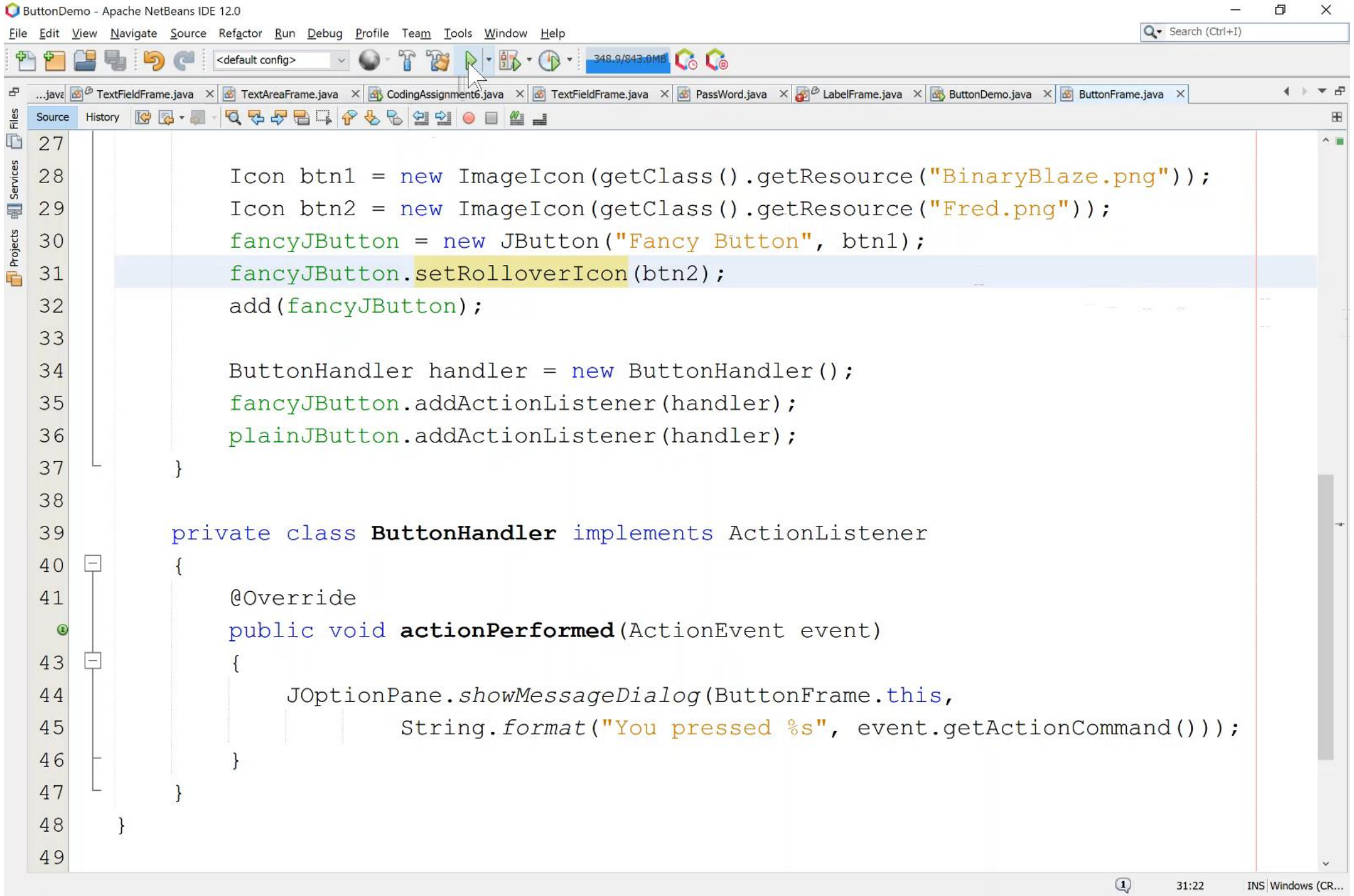
The text on the face of the button is called a **button label**.

Event handling for buttons is very similar to labels.

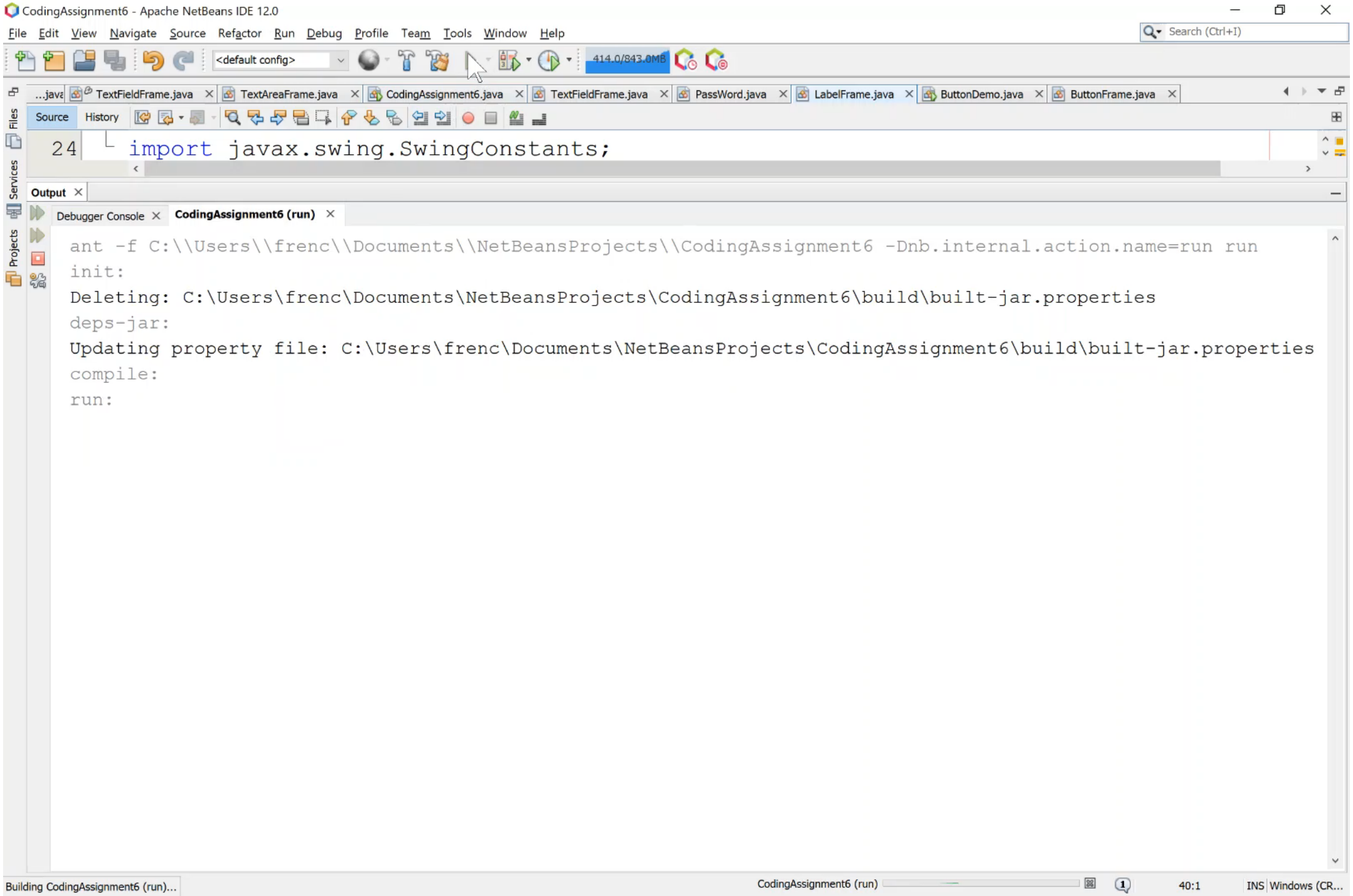
```
package buttondemo;

import javax.swing.JFrame;

public class ButtonDemo
{
    public static void main(String[] args)
    {
        ButtonFrame buttonFrame = new ButtonFrame();
        buttonFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        buttonFrame.setSize(675, 410);
        buttonFrame.setVisible(true);
    }
}
```

```
public class ButtonFrame extends JFrame
{
    private final JButton plainJButton;
    private final JButton fancyJButton;

    public ButtonFrame()
    {
        super("Testing Buttons");
        setLayout(new FlowLayout());

        plainJButton = new JButton("Plain Button");
        add(plainJButton);

        Icon btn1 = new ImageIcon(getClass().getResource("BinaryBlaze.png"));
        Icon btn2 = new ImageIcon(getClass().getResource("Fred.png"));
        fancyJButton = new JButton("Fancy Button", btn1);
        fancyJButton.setRolloverIcon(btn2);
        add(fancyJButton);

        ButtonHandler handler = new ButtonHandler();
        fancyJButton.addActionListener(handler);
        plainJButton.addActionListener(handler);
    }

    private class ButtonHandler implements ActionListener
    {
        @Override
        public void actionPerformed(ActionEvent event)
        {
            JOptionPane.showMessageDialog(ButtonFrame.this,
                    String.format("You pressed %s", event.getActionCommand()));
        }
    }
}
```


ButtonDemo - Apache NetBeans IDE 12.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
667.0/843.0MB
TextFieldFrame.java
TextAreaFrame.java
CodingAssignment6.java
TextFieldFrame.java
PassWord.java
LabelFrame.java
ButtonDemo.java
ButtonFrame.java
Source History
Files Services Projects
27 Icon btn1 = new ImageIcon(getClass().getResource("BinaryBlaze.png"));
28 Icon btn2 = new ImageIcon(getClass().getResource("Fred.png"));
29 fancyJButton = new JButton("Fancy Button", btn1);
30 fancyJButton.setRolloverIcon(btn2);
31 add(fancyJButton);
32
33 ButtonHandler handler = new ButtonHandler();
35 fancyJButton.addActionListener(handler);
36 plainJButton.addActionListener(handler);
37 }
38
39 private class ButtonHandler implements ActionListener
40 {
41 @Override
public void actionPerformed(ActionEvent event)
43 {
44 JOptionPane.showMessageDialog(ButtonFrame.this,
45 String.format("You pressed %s", event.getActionCommand()));
46 }
47 }
48 }
49
34:24
INS Windows (CR...


ButtonDemo - Apache NetBeans IDE 12.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
348.9/843.0MB
TextFieldFrame.java
TextAreaFrame.java
CodingAssignment6.java
TextFieldFrame.java
PassWord.java
LabelFrame.java
ButtonDemo.java
ButtonFrame.java
Source History
Files
Services
Projects
27 Icon btn1 = new ImageIcon(getClass().getResource("BinaryBlaze.png"));
28 Icon btn2 = new ImageIcon(getClass().getResource("Fred.png"));
29 fancyJButton = new JButton("Fancy Button", btn1);
30 fancyJButton.setRolloverIcon(btn2);
31 add(fancyJButton);
32
33 ButtonHandler handler = new ButtonHandler();
34 fancyJButton.addActionListener(handler);
35 plainJButton.addActionListener(handler);
36 }
37
38 private class ButtonHandler implements ActionListener
39 {
40 @Override
41 public void actionPerformed(ActionEvent event)
43 {
44 JOptionPane.showMessageDialog(ButtonFrame.this,
45 String.format("You pressed %s", event.getActionCommand()));
46 }
47 }
48 }
49
31:22
INS Windows (CR...

Coding Assignment 6


CodingAssignment6 - Apache NetBeans IDE 12.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
414.0/843.0MB
TextFieldFrame.java
TextAreaFrame.java
CodingAssignment6.java
TextFieldFrame.java
PassWord.java
LabelFrame.java
ButtonDemo.java
ButtonFrame.java
Source History
24 import javax.swing.SwingConstants;
Output
Debugger Console
CodingAssignment6 (run)
ant -f C:\\Users\\frenc\\Documents\\NetBeansProjects\\CodingAssignment6 -Dnb.internal.action.name=run run
init:
Deleting: C:\Users\frenc\Documents\NetBeansProjects\CodingAssignment6\build\built-jar.properties
deps-jar:
Updating property file: C:\Users\frenc\Documents\NetBeansProjects\CodingAssignment6\build\built-jar.properties
compile:
run:
Files
Services
Projects
Building CodingAssignment6 (run)...
CodingAssignment6 (run)
40:1
INS Windows (CR...

Coding Assignment 6

Pick at least 4 related images. My app uses 4 characters from the cartoon series Flintstones.

Fred.png

Barney.png

Wilma.png

Betty.png

You can use .png or .jpeg or .gif files. Your file names should match the string you want to be entered as the guess for the character's name. Store these files in the same directory as your class files.

Coding Assignment 6

GUI - Label

```
package labeltest;

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(460,380);
        labelFrame.setVisible(true);
    }
}
```

Create your `Code6_xxxxxxxxxx` project.

In `main()`, instantiate an object of the class `Password`.

Setup your `password` object the same way we set `labelFrame` in class `LabelTest` in the slides.

You will need to adjust the size as needed for your project.

This will be the only code that goes in `main()` and in class `Code6_xxxxxxxxxx`

```
public class TextFieldFrame extends JFrame
{
    private final JTextField textField1;
    private final JTextField textField2;
    private final JTextField textField3;
    private final JPasswordField passwordField;
```

Coding Assignment 6

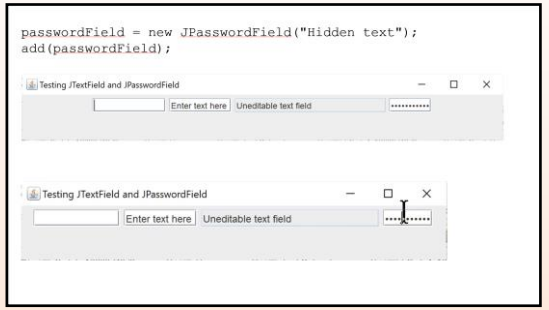
Create a `Password` class file.

Class `Password` extends `JFrame`

Set up a `private final JPassword` variable like we did in class.

Create a variable to hold the user entered password.

Create a `final String` variable and set it to the actual password.



Coding Assignment 6

Create a `Password` class file.

Create the `Password` class constructor

- Call the superclass constructor
- set the layout to `FlowLayout`
- instantiate a `JPasswordField`
- look in the Java documentation on how to **`set`** the **`E`**cho **`char`** to `'X'`
- add

```
TextFieldHandler handler = new TextFieldHandler();
textField1.addActionListener(handler);
textField2.addActionListener(handler);
textField3.addActionListener(handler);
passwordField.addActionListener(handler);
```

Coding Assignment 6

Add event handling to `Password` class

Instantiate an event handler from class `EventHandler`

Add an `ActionListener` to `passwordField`

Coding Assignment 6

```
private class TextFieldHandler implements ActionListener
{
    @Override
    public void actionPerformed(ActionEvent event)
    {
        String string = "";

        if (event.getSource() == textField1)
            string = String.format("textField1: %s", event.getActionCommand());
        else if (event.getSource() == textField2)
            string = String.format("textField2: %s", event.getActionCommand());
        else if (event.getSource() == textField3)
            string = String.format("textField3: %s", event.getActionCommand());
        else if (event.getSource() == passwordField)
            string = String.format("passwordField: %s", event.getActionCommand());

        JOptionPane.showMessageDialog(null, string);
    }
}
```

Add inner class `EventHandler`

private inner class `EventHandler` will implement interface `ActionListener`

If `event.getSource()` is `passwordField`, then set user entered password to `event.getActionCommand()`.

If user entered password equals the actual password, then pass `false` to `setVisible()` to hide the password entry box and set up `GameFrame` (next slide).

If user entered password does not equal the actual password, then use a `MessageDialog` box to display invalid password message.

Coding Assignment 6

GUI - Label

```
package labeltest;

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(460,380);
        labelFrame.setVisible(true);
    }
}
```

Instantiating `GameFrame` inside `EventHandler`

Set up `GameFrame` like you did `Password`.

- instantiate it
- set default close operation
- set size
- set visibile

Now create a new class file named `GameFrame`.

Coding Assignment 6

Create class `GameFrame`

`GameFrame` extends `JFrame`

Declare 4 `private final` variables

```
JLabel label1
JButton OKButton
JButton CancelButton
JTextField textField1
```

Declare a `String` to hold the character's name/answer to the question. I called mine `CCName`.

Coding Assignment 6

`GameFrame` constructor

Call super constructor, set the layout and set the default close operation.

Use a random number to choose one of your four character names. I used a `switch` with the random number to pick 1 of my 4. For example, a random number of 2 sets my `CCName` variable to "`Wilma`".

Use the randomly chosen name to create the `Icon` object.

```
Icon CC = new ImageIcon(getClass().getResource(CCName + ".png"));
```

Coding Assignment 6

```
public LabelFrame()
{
    super("Testing JLabel");
    setLayout(new FlowLayout());

    label1 = new JLabel("Label with text");
    label1.setToolTipText("This is label1");
    add(label1);

    Icon BB = new ImageIcon(getClass().getResource("BinaryBlaze.png"));
    label2 = new JLabel("Label with text and icon", BB, SwingConstants.LEFT);
    label2.setToolTipText("This is label2");
    add(label2);

    label3 = new JLabel();
    label3.setText("Label with icon and text at bottom");
    label3.setIcon(BB);
    label3.setHorizontalTextPosition(SwingConstants.CENTER);
    label3.setVerticalTextPosition(SwingConstants.BOTTOM);
    label3.setToolTipText("This is label3");
    add(label3);
}
```

`GameFrame` constructor continued…

Create a new `JLabel` called `label1`

- set the text
- set the icon
- set to horizontal and vertical position
- set the tool tip to be a hint
- add it

Coding Assignment 6

`GameFrame` constructor continued…

Instantiate an event handler from class `EventHandler` and name it `handler`

We'll create the inner class `EventHandler` shortly…

NOTE : all components in the `GameFrame` class will use the same event handler – do not create separate ones for the text field and the buttons.

```
public TextFieldFrame()
{
    super("Testing JTextField and JPasswordField");
    setLayout(new FlowLayout());

    textField1 = new JTextField(10);
    add(textField1);

    textField2 = new JTextField("Enter text here ");
    add(textField2);

    textField3 = new JTextField("Uneditable text field", 21);
    textField3.setEditable(false);
    add(textField3);

    passwordField = new JPasswordField("Hidden text");
    add(passwordField);

    TextFieldHandler handler = new TextFieldHandler();
    textField1.addActionListener(handler);
    textField2.addActionListener(handler);
    textField3.addActionListener(handler);
    passwordField.addActionListener(handler);
}
```

Coding Assignment 6

`GameFrame` constructor continued…

Instantiate a new `JTextField` named `textField1`.

Check the Java documentation on how to **`selectAll`** of the text in the field so that you can just start typing and overwrite it without having to manual select it or delete it.

Set an `ActionListener` for `textField1` and add `textField1` to the container.

Coding Assignment 6

```
public class ButtonFrame extends JFrame
{
    private final JButton plainJButton;
    private final JButton fancyJButton;

    public ButtonFrame()
    {
        super("Testing Buttons");
        setLayout(new FlowLayout());

        plainJButton = new JButton("Plain Button");
        add(plainJButton);

        Icon btn1 = new ImageIcon(getClass().getResource("BinaryBlaze.png"));
        Icon btn2 = new ImageIcon(getClass().getResource("Fred.png"));
        fancyJButton = new JButton("Fancy Button", btn1);
        fancyJButton.setRolloverIcon(btn2);
        add(fancyJButton);

        ButtonHandler handler = new ButtonHandler();
        fancyJButton.addActionListener(handler);
        plainJButton.addActionListener(handler);
    }

    private class ButtonHandler implements ActionListener
    {
        @Override
        public void actionPerformed(ActionEvent event)
        {
            JOptionPane.showMessageDialog(ButtonFrame.this,
                    String.format("You pressed %s", event.getActionCommand()));
        }
    }
}
```

`GameFrame` constructor continued and finished

Instantiate a new `JButton` named `OKButton`.

Set an `ActionListener` for `OKButton` and add `OKButton` to the container.

Instantiate a new `JButton` named `CancelButton`.

Set an `ActionListener` for `CancelButton` and add `CancelButton` to the container.

```
private class TextFieldHandler implements ActionListener
{
    @Override
    public void actionPerformed(ActionEvent event)
    {
        String string = "";

        if (event.getSource() == textField1)
            string = String.format("textField1: %s", event.getActionCommand());
        else if (event.getSource() == textField2)
            string = String.format("textField2: %s", event.getActionCommand());
        else if (event.getSource() == textField3)
            string = String.format("textField3: %s", event.getActionCommand());
        else if (event.getSource() == passwordField)
            string = String.format("passwordField: %s", event.getActionCommand());

        JOptionPane.showMessageDialog(null, string);
    }
}
```

Coding Assignment 6

Create inner class `EventHandler`

private inner class `EventHandler` will implement interface `ActionListener`

Set up a `String` that will hold the response that you will show to the user

Create a `boolean` that will be true if the user entered guess matches the character's name – initialize it to false

```
private class TextFieldHandler implements ActionListener
{
    @Override
    public void actionPerformed(ActionEvent event)
    {
        String string = "";

        if (event.getSource() == textField1)
            string = String.format("textField1: %s", event.getActionCommand());
        else if (event.getSource() == textField2)
            string = String.format("textField2: %s", event.getActionCommand());
        else if (event.getSource() == textField3)
            string = String.format("textField3: %s", event.getActionCommand());
        else if (event.getSource() == passwordField)
            string = String.format("passwordField: %s", event.getActionCommand());

        JOptionPane.showMessageDialog(null, string);
    }
}
```

Coding Assignment 6

Create inner class `EventHandler`

if the character's name equals* the user entered string

    set the response string to the guessed correctly phrase

    set the `boolean` to true

else

    set the response string to the guessed incorrectly phrase

*be sure to ignore case

Coding Assignment 6

```
private class TextFieldHandler implements ActionListener
{
    @Override
    public void actionPerformed(ActionEvent event)
    {
        String string = "";

        if (event.getSource() == textField1)
            string = String.format("textField1: %s", event.getActionCommand());
        else if (event.getSource() == textField2)
            string = String.format("textField2: %s", event.getActionCommand());
        else if (event.getSource() == textField3)
            string = String.format("textField3: %s", event.getActionCommand());
        else if (event.getSource() == passwordField)
            string = String.format("passwordField: %s", event.getActionCommand());

        JOptionPane.showMessageDialog(null, string);
    }
}
```

Create inner class `EventHandler`

if `event.getSource()` is the `OKButton` or `textField1`

show a `MessageDialog` box with the response string

if your `boolean` is true

use `System.exit(0)` to shut down the program

else if `event.getSource()` is the `CancelButton`

use `System.exit(0)` to shut down the program

Coding Assignment 6

Almost everything you need for the assignment is in the Swing slides. There are a few minor things I want you to find on your own in the online documentation in order to become familiar with it.

https://docs.oracle.com/javase/7/docs/api/allclasses-noframe.html

Google and Stack Overflow are NOT your friends for this assignment – they will lead you down a complicated path that will waste your time.

Stick with the simple things we did in class for this assignment.

Coding Assignment 6

Create a standalone executable of your program by creating a `.jar` file.

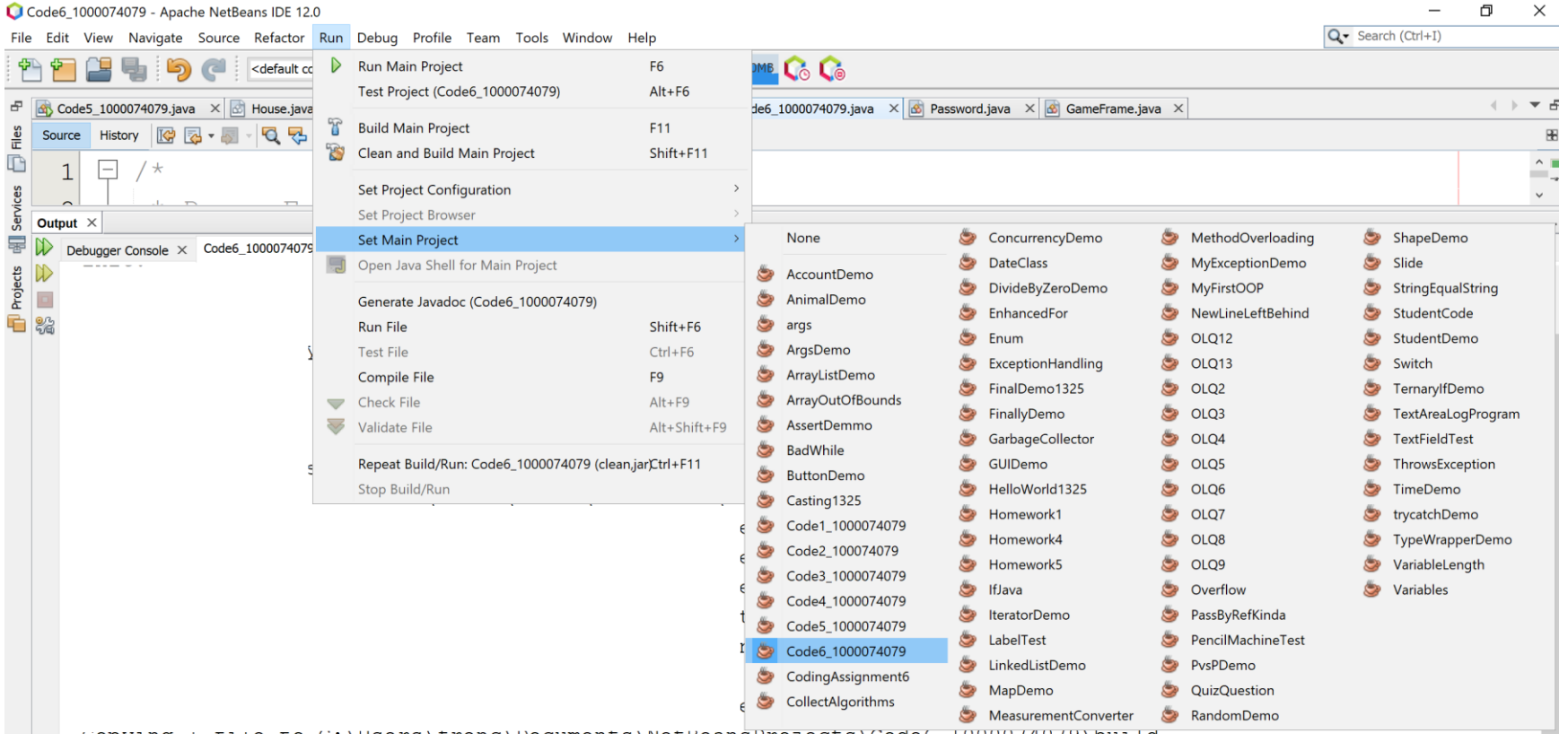
I have included the directions for creating a `.jar` file in NetBeans.

If you are using a different IDE, then look for similar terms.

Your code will be graded by running the executable file and looking at your code.

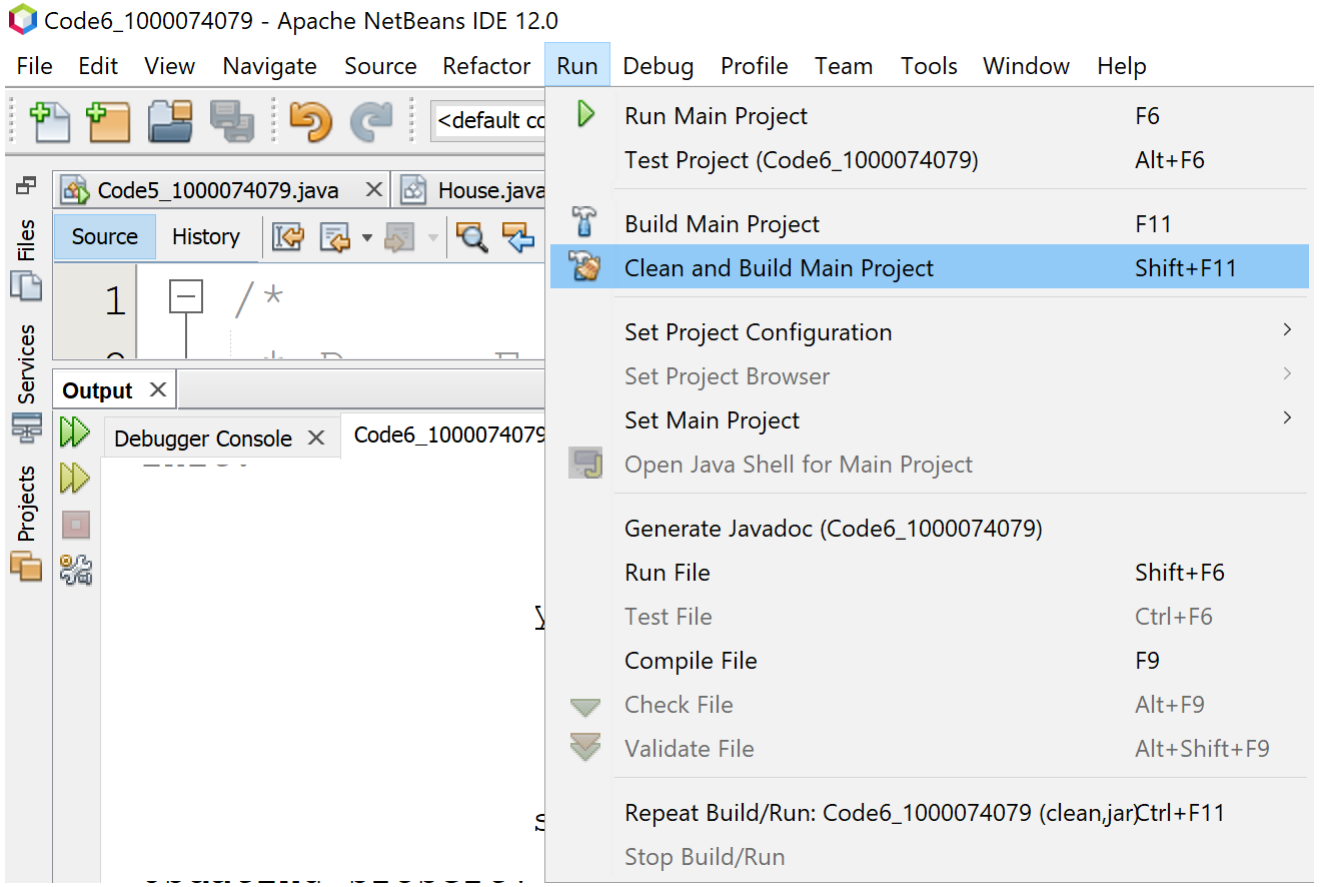
Coding Assignment 6

Under `Run`, go to `Set Main Project` and make sure `Code6_xxxxxxxxx` is set as the Main Project.



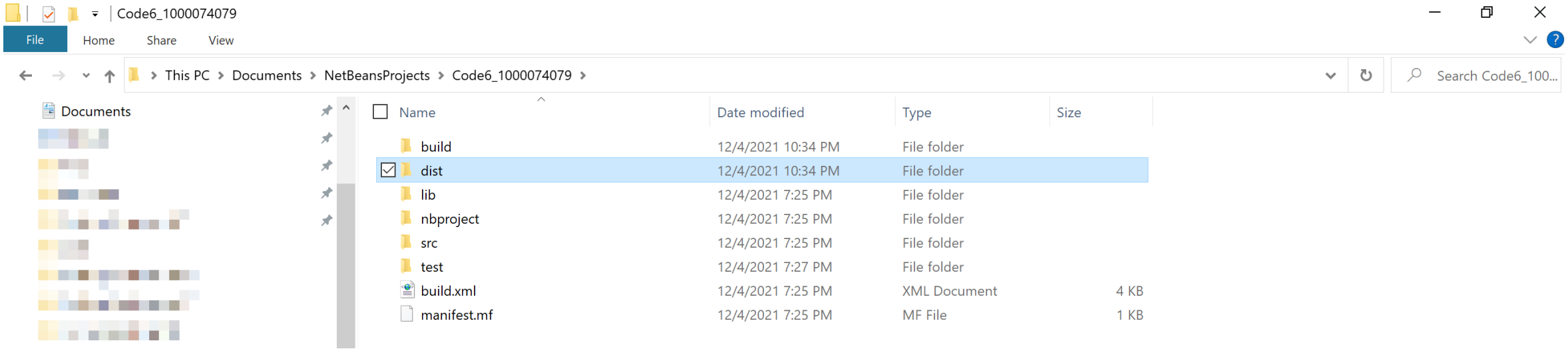
Coding Assignment 6

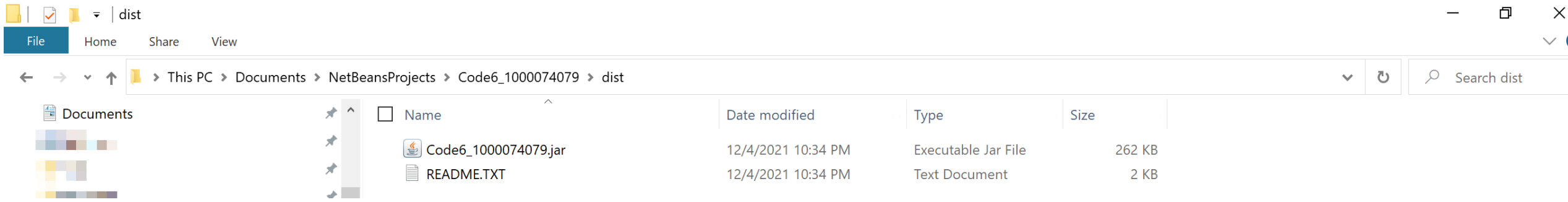
Under `Run`**, click on** `Clean and Build Main Project`



Coding Assignment 6

The `Code6_xxxxxxxxxx.jar` file will be created in a folder named `dist` in your NetBeans project folder.





You pick the password.

You pick the images.

You pick the question.

You pick the answer(s).

You pick the text of the `MessageDialog` boxes for a right and a wrong answer.

Your application needs to behave in the same way as the example in the assignment. You do not need to have the exact layout as my example. I used a vertical box and a horizontal box to control the placement of the components but you are not required to do this.

Please be professional with your choices – no offensive images or offensive language.