Lecture 3 - Java Graphical User Interface (GUI): Java AWT

Guiliang Liu

The Chinese University of Hong Kong, Shenzhen

CSC-1004: Computational Laboratory Using Java
Course Page: [Click]

Outline

- Java Abstract Window Toolkit (AWT)
- Java Swing
- Java FX

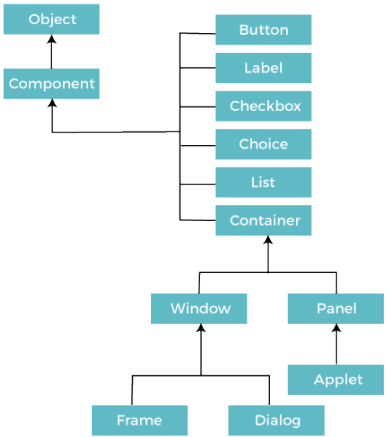
Java AWT

Java AWT is an API to develop Graphical User Interface (GUI) in Java.

- Java AWT components are platform-dependent i.e. components are displayed according to the view of the operating system.
- AWT is heavyweight i.e. its components are using the resources of the underlying operating system (OS).

Java AWT

Java AWT is an API to develop Graphical User Interface (GUI) in Java.



- **Components**: All the elements like the button, text fields, scroll bars, etc. In order to place every component in a particular position on a screen, we need to add them to a container.
- **Container**: The Container is a component in AWT that can contain other components like buttons, text fields, labels, etc.

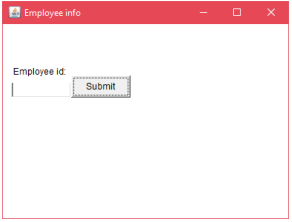
Java AWT

Types of containers:

- **Window**. The window is the container that has no borders and menu bars. You must use a frame, dialog, or another window for creating a window. We need to create an instance of the Window class to create this container.
- **Panel**. The Panel is the container that doesn't contain a title bar, border, or menu bar. It is a generic container for holding the components.
- **Frame**. The Frame is the container that contains a title bar and border and can have menu bars. The frame is the most widely used container while developing an AWT application.

Java AWT Example

```
// class AWTExample2 directly creates instance of Frame class
class AWTExample2 {

  // initializing using constructor
  AWTExample2() {

    // creating a Frame
    Frame f = new Frame();

    // creating a Label
    Label l = new Label("Employee id:");

    // creating a Button
    Button b = new Button("Submit");

    // creating a TextField
    TextField t = new TextField();

    // setting position of above components in the frame
    l.setBounds(20, 80, 80, 30);
    t.setBounds(20, 100, 80, 30);
    b.setBounds(100, 100, 80, 30);

    // adding components into frame
    f.add(b);
    f.add(l);
    f.add(t);

    // frame size 300 width and 300 height
    f.setSize(400,300);

    // setting the title of frame
    f.setTitle("Employee info");

    // no layout
    f.setLayout(null);

    // setting visibility of frame
    f.setVisible(true);
}
```



Java Event Handling

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc.

```
class AEvent extends Frame implements ActionListener{
TextField tf;
AEvent(){

//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);

//register listener
b.addActionListener(this);//passing current instance
```

```
//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
public static void main(String args[]){
new AEvent();
}
}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java Event Handling

To perform event handling, **we must register the component with the Listener**.

Many classes provide registration methods. For example:

- **Button**: public void addActionListener(ActionListener a)
- **MenuItem**: public void addActionListener(ActionListener a)
- **TextField**: public void addActionListener(ActionListener a) and public void addTextListener(TextListener a)
- **TextArea**: public void addTextListener(TextListener a)
- **Checkbox**: public void addItemListener(ItemListener a)
- **Choice**: public void addItemListener(ItemListener a)
- **List**: public void addActionListener(ActionListener a) and public void addItemListener(ItemListener a)

Java AWT Button

- A button is basically a control component with a label.
- The application result in some action (event) when the button is pushed.
- To perform an action on a button being pressed and released, the ActionListener interface needs to be implemented. The registered new listener can receive events from the button by calling addActionListener method of the button

Java AWT Button Example

In the following example, we are handling the button click events by implementing ActionListener Interface.

```
class AEvent extends Frame implements ActionListener{
TextField tf;
AEvent(){

//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);

//register listener
b.addActionListener(this);//passing current instance
```

```
//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
public static void main(String args[]){
new AEvent();
}
}
```

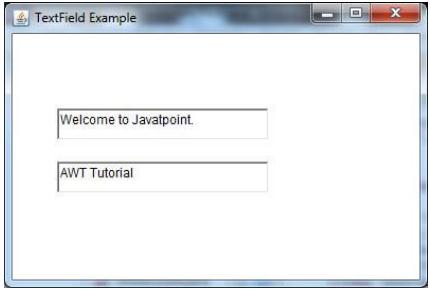
Java AWT Label

- The object of the Label class is a component for placing text in a container.
- It is used to display a single line of read only text.
- The text can be changed by a programmer but a user cannot edit it directly.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java AWT TextField

- The object of a TextField class is a text component that allows a user to enter a single line of text and edit it.

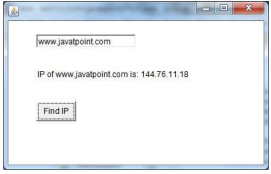


香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java AWT TextField and Label Example

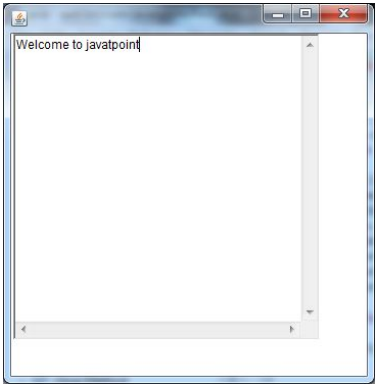
In the following example, we are creating the objects of TextField, Label and Button classes and adding them to the Frame. When we add the website in the text field and click on the button, we get the IP address of website.

```
// creating class which implements ActionListener interface and inherits Frame class
public class LabelExample2 extends Frame implements ActionListener{

  // creating objects of TextField, Label and Button class
  TextField tf;
  Label l;
  Button b;

  // constructor to instantiate the above objects
  LabelExample2() {
    tf = new TextField();
    tf.setBounds(50, 50, 150, 20);

    l = new Label();
    l.setBounds(50, 100, 250, 20);

    b = new Button("Find IP");
    b.setBounds(50,150,60,30);
    b.addActionListener(this);
    add(b);
    add(tf);
    add(l);

    setSize(400,400);
    setLayout(null);
    setVisible(true);
  }

  // defining actionPerformed method to generate an event
  public void actionPerformed(ActionEvent e) {
    try {
    String host = tf.getText();
    String ip = java.net.InetAddress.getByName(host).getHostAddress();
    l.setText("IP of "+host+" is: "+ip);
    }
    catch (Exception ex) {
      System.out.println(ex);
    }
  }
```



Java AWT TextArea

- The object of a TextArea class is a multiline region that displays text.
- It allows the editing of multiple-line text. The text area allows us to type as much text as we want.

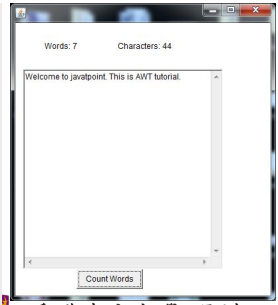


Java AWT TextArea Example

The following example displays a text area in the frame where it extends the Frame class and implements ActionListener interface, where we are counting the number of characters and words entered in the text area.

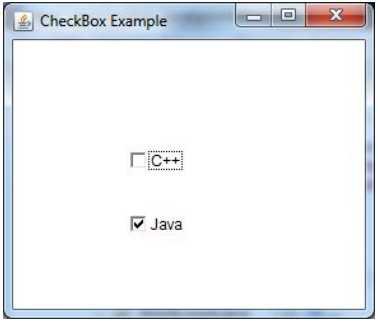
```
public class TextAreaExample2 extends Frame implements ActionListener {
// creating objects of Label, TextArea and Button class.
Label l1, l2;
TextArea area;
Button b;
// constructor to instantiate
TextAreaExample2() {
// instantiating and setting the location of components on the frame
   l1 = new Label();
   l1.setBounds(50, 50, 100, 30);
   l2 = new Label();
   l2.setBounds(160, 50, 100, 30);
   area = new TextArea();
   area.setBounds(20, 100, 300, 300);
   b = new Button("Count Words");
   b.setBounds(100, 400, 100, 30);

// adding ActionListener to button
   b.addActionListener(this);
```

```
// adding components to frame
   add(l1);
add(l2);
add(area);
add(b);
// setting the size, layout and visibility of frame
   setSize(400, 450);
   setLayout(null);
   setVisible(true);
}
// generating event text area to count number of words and characters
public void actionPerformed(ActionEvent e) {
   String text = area.getText();
   String words[]=text.split("\\s");
   l1.setText("Words: "+words.length);
   l2.setText("Characters: "+text.length());
}
```



Java AWT Checkbox

The Checkbox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a Checkbox changes its state from "on" to "off" or from "off" to "on".

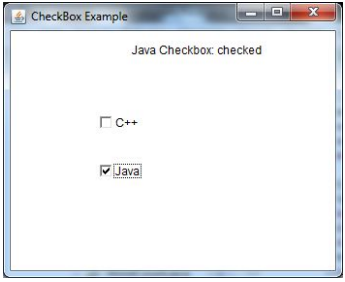


Java AWT Checkbox Example

This example creates two checkboxes and adds them to the Frame. We add the ItemListener with the checkbox which displays the state of the checkbox.

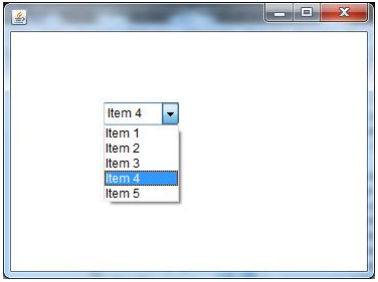
```
import java.awt.*;
import java.awt.event.*;
public class CheckboxExample2
{
// constructor to initialize
    CheckboxExample2() {
// creating the frame
      Frame f = new Frame ("CheckBox Example");
// creating the label
      final Label label = new Label();
// setting the alignment, size of label
  label.setAlignment(Label.CENTER);
      label.setSize(400,100);
// creating the checkboxes
      Checkbox checkbox1 = new Checkbox("C++");
      checkbox1.setBounds(100, 100,  50, 50);
      Checkbox checkbox2 = new Checkbox("Java");
      checkbox2.setBounds(100, 150,  50, 50);
// adding the checkbox to frame
f.add(checkbox1);
f.add(checkbox2);
f.add(label);
```

```
      checkbox1.addItemListener(new ItemListener() {
          public void itemStateChanged(ItemEvent e) {
            label.setText("C++ Checkbox: "
            + (e.getStateChange()==1?"checked":"unchecked"));
          }
        });
      checkbox2.addItemListener(new ItemListener() {
          public void itemStateChanged(ItemEvent e) {
            label.setText("Java Checkbox: "
            + (e.getStateChange()==1?"checked":"unchecked"));
          }
        });
// setting size, layout and visibility of frame
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
    }
// main method
public static void main(String args[])
{
   new CheckboxExample2();
}
}
```



Java AWT Choice

The object of the Choice class is used to show a popup menu of choices. The choice selected by the user is shown at the top of the menu.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java AWT Choice Example with ActionListener

This example creates a choice menu with 5 items. We create a button and a label.

Here, we add an event to the button component using addActionListener() method.

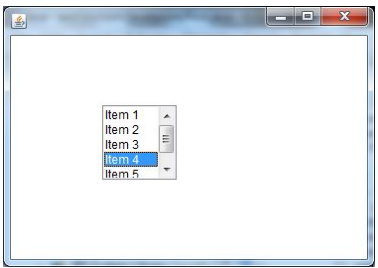
```
public class ChoiceExample2 {

    // class constructor
    ChoiceExample2() {

    // creating a frame
    Frame f = new Frame();

    // creating a final object of Label class
    final Label label = new Label();

    // setting alignment and size of label component
    label.setAlignment(Label.CENTER);
    label.setSize(400, 100);

    // creating a button
    Button b = new Button("Show");

    // setting the bounds of button
    b.setBounds(200, 100, 50, 20);
```

```
// creating final object of Choice class
final Choice c = new Choice();

// setting bounds of choice menu
c.setBounds(100, 100, 75, 75);

// adding 5 items to choice menu
c.add("C");
c.add("C++");
c.add("Java");
c.add("PHP");
c.add("Android");

// adding above components into the frame
f.add(c);
f.add(label);
f.add(b);

// setting size, layout and visibility of frame
f.setSize(400, 400);
f.setLayout(null);
```

```
// adding event to the button
// which displays the selected item from the list when button is clicked
b.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
 String data = "Programming language Selected: "+ c.getItem(c.getSelectedIndex());
 label.setText(data);
}
});
}
```

Java AWT List

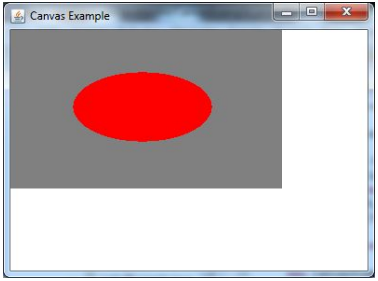
The object of the List class represents a list of text items. With the help of the List class, the user can choose either one item or multiple items. It inherits the Component class.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java AWT Canvas

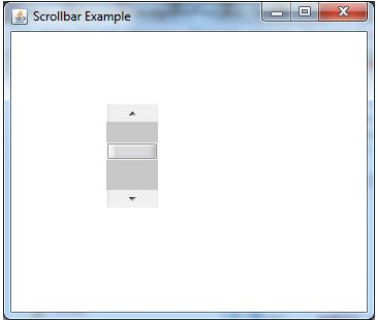
The Canvas class controls and represents a blank rectangular area where the application can draw or trap input events from the user. It inherits the Component class.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java AWT Scrollbar

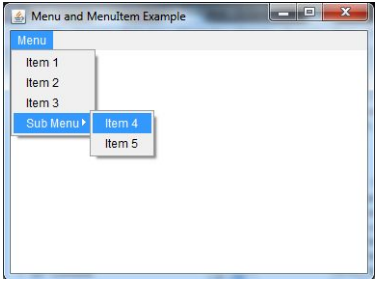
The object of the Scrollbar class is used to add a horizontal and vertical scrollbar. The scrollbar is a GUI component that allows us to see the invisible number of rows and columns.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java AWT MenuItem and Menu

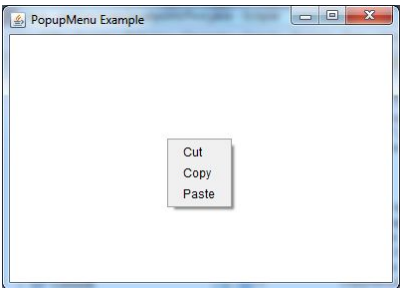
The object of MenuItem class adds a simple labeled menu item on the menu. The items used in a menu must belong to the MenuItem or any of its subclass. The object of the Menu class is a pull-down menu component which is displayed on the menu bar. It inherits the MenuItem class.

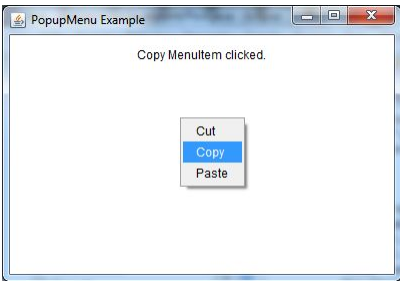


香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java AWT PopupMenu

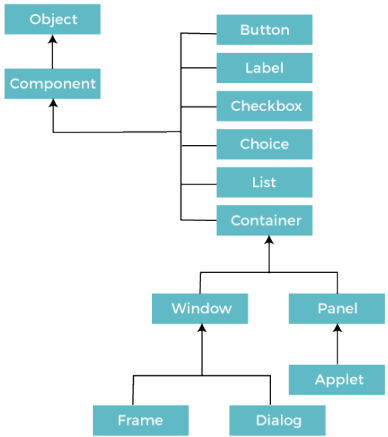
PopupMenu can be dynamically popped up at specific position within a component. It inherits the Menu class.





Java AWT

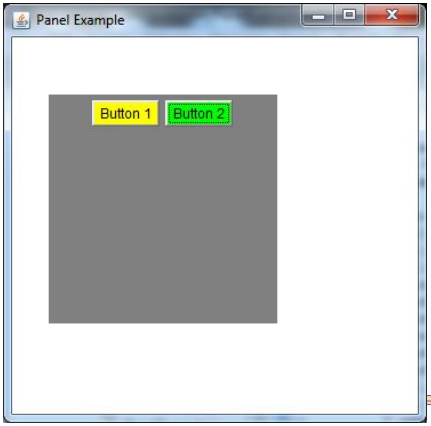
Java AWT is an API to develop Graphical User Interface (GUI) in Java.



- **Components**: All the elements like the button, text fields, scroll bars, etc. In order to place every component in a particular position on a screen, we need to add them to a container.
- **Container**: The Container is a component in AWT that can contain other components like buttons, text fields, labels, etc.

Java AWT Panel

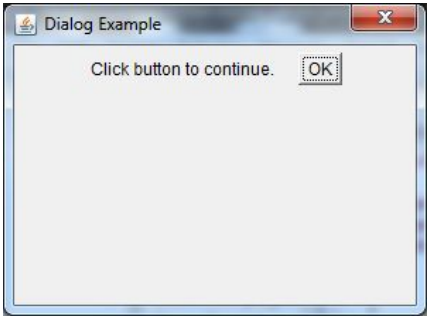
The Panel is the simplest container class. It provides space in which an application can attach any other component. It inherits the Container class.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java AWT Dialog

The Dialog control represents a top-level window with a border and a title used to take some form of input from the user. It inherits the Window class. Unlike Frame, it doesn't have to maximize and minimize buttons.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java Event Handling

To perform event handling, **we must register the component with the Listener**.

Many classes provide registration methods. For example:

- **Button**: public void addActionListener(ActionListener a)
- **MenuItem**: public void addActionListener(ActionListener a)
- **TextField**: public void addActionListener(ActionListener a) and public void addTextListener(TextListener a)
- **TextArea**: public void addTextListener(TextListener a)
- **Checkbox**: public void addItemListener(ItemListener a)
- **Choice**: public void addItemListener(ItemListener a)
- **List**: public void addActionListener(ActionListener a) and public void addItemListener(ItemListener a)

Java ActionListener

The Java ActionListener is notified whenever you click on the button or menu item. If you implement the ActionListener class, you need to follow 3 steps:

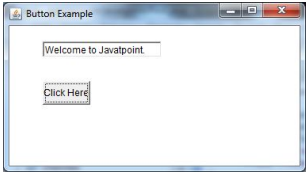
1. Implement the ActionListener interface in the class.
2. Register the component with the Listener.
3. Override the actionPerformed() method.

Java ActionListener

The Java ActionListener is notified whenever you click on the button or menu item. If you implement the ActionListener class, you need to follow 3 steps:

```
import java.awt.*;
import java.awt.event.*;
//1st step
public class ActionListenerExample implements ActionListener{
public static void main(String[] args) {
    Frame f=new Frame("ActionListener Example");
    final TextField tf=new TextField();
    tf.setBounds(50,50, 150,20);
    Button b=new Button("Click Here");
    b.setBounds(50,100,60,30);
```

```
    //2nd step
    b.addActionListener(this);
    f.add(b);f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
//3rd step
public void actionPerformed(ActionEvent e){
        tf.setText("Welcome to Javatpoint.");
}
}
```



Java MouseListener

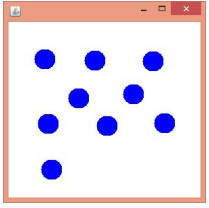
The Java MouseListener is notified whenever you change the state of the mouse. It has five methods.

1. public abstract void mouseClicked(MouseEvent e);
2. public abstract void mouseEntered(MouseEvent e);
3. public abstract void mouseExited(MouseEvent e);
4. public abstract void mousePressed(MouseEvent e);
5. public abstract void mouseReleased(MouseEvent e);

Java MouseListener

The Java MouseListener is notified whenever you change the state of the mouse. It has five methods.

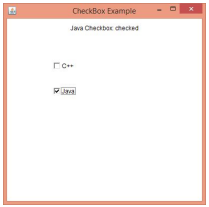
```
public class MouseListenerExample2 extends Frame implements MouseListener{
  MouseListenerExample2(){
    addMouseListener(this);

    setSize(300,300);
    setLayout(null);
    setVisible(true);
  }
  public void mouseClicked(MouseEvent e) {
    Graphics g=getGraphics();
    g.setColor(Color.BLUE);
    g.fillOval(e.getX(),e.getY(),30,30);
  }
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {}
  public void mouseReleased(MouseEvent e) {}

public static void main(String[] args) {
  new MouseListenerExample2();
}
}
```



Java ItemListener

The Java ItemListener is notified whenever you click on the checkbox. It has only one method: itemStateChanged().

```
public class ItemListenerExample implements ItemListener{
  Checkbox checkBox1,checkBox2;
  Label label;
  ItemListenerExample(){
    Frame f= new Frame("CheckBox Example");
    label = new Label();
    label.setAlignment(Label.CENTER);
    label.setSize(400,100);
    checkBox1 = new Checkbox("C++");
    checkBox1.setBounds(100,100, 50,50);
    checkBox2 = new Checkbox("Java");
    checkBox2.setBounds(100,150, 50,50);
    f.add(checkBox1); f.add(checkBox2); f.add(label);
    checkBox1.addItemListener(this);
    checkBox2.addItemListener(this);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
  }
  public void itemStateChanged(ItemEvent e) {
    if(e.getSource()==checkBox1)
      label.setText("C++ Checkbox: "
      + (e.getStateChange()==1?"checked":"unchecked"));
    if(e.getSource()==checkBox2)
    label.setText("Java Checkbox: "
    + (e.getStateChange()==1?"checked":"unchecked"));
  }
public static void main(String args[])
{
  new ItemListenerExample();
}
}
```



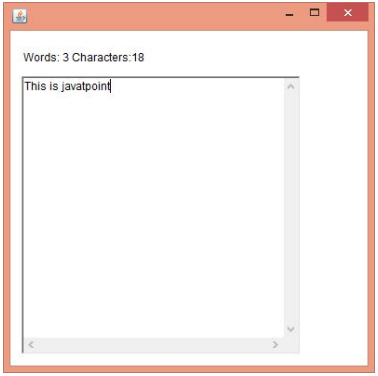
Java KeyListener

The Java KeyListener is notified whenever you change the state of key. It has three methods.

1. public abstract void keyPressed (KeyEvent e), invoked when a key is pressed.
2. public abstract void keyReleased (KeyEvent e), invoked when a key is released.
3. public abstract void keyTyped (KeyEvent e), invoked when a key is typed.

Java KeyListener

The Java KeyListener is notified whenever you change the state of key. It has three methods.



Java WindowListener

The WindowListener is notified whenever you change the state of the window by:

1. public abstract void windowActivated (WindowEvent e), invoked when the Window is set to active.
2. public abstract void windowClosed (WindowEvent e), invoked when a window has been closed.
3. public abstract void windowClosing (WindowEvent e), invoked when the user attempts to close the window from the system menu.

Java WindowListener

The WindowListener is notified whenever you change the state of the window by:

1. public abstract void windowDeactivated (WindowEvent e);, invoked when a Window is not an active Window anymore.
2. public abstract void windowDeiconified (WindowEvent e), invoked when a window is changed from a minimized to a normal state.
3. public abstract void windowIconified (WindowEvent e), invoked when a window is changed from a normal to a minimized state.
4. public abstract void windowOpened (WindowEvent e), invoked when a window is made visible for the first time.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen