THE UNIVERSITY OF AUCKLAND

COMPSCI 230

Software Design and Construction

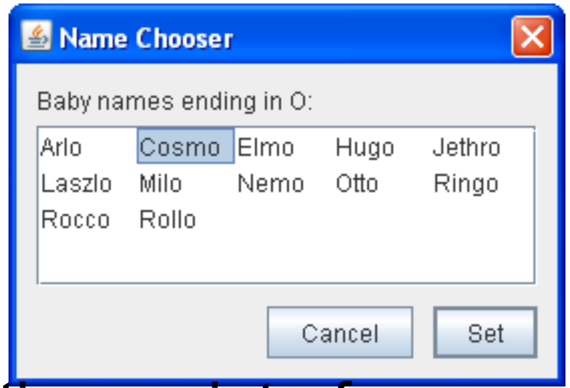
Swing 1

2013-04-17

Recap: Swing Design Principles

1. GUI is built as **containment hierarchy** of widgets (i.e. the parent-child nesting relation between them)

2. Event objects and event listeners
   - **Event object**: is created when event occurs (e.g. click), contains additional info (e.g. mouse coordinates)
   - **Event listener**: object implementing an interface with an event handler method that gets an event object as argument



3. Separation of Model and View:
   - **Model**: the data that is presented by a widget
   - **View**: the actual presentation on the screen

Recap: Swing Hello World

THE UNIVERSITY OF AUCKLAND

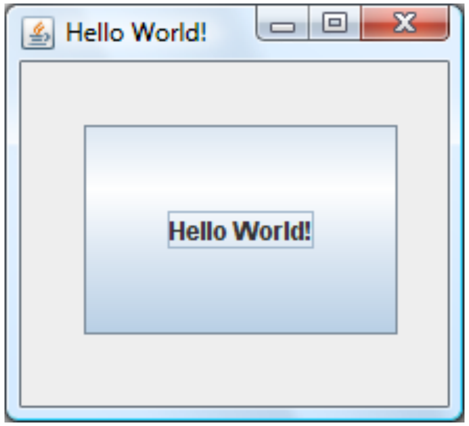
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class HelloWorld {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Hello World!");
        frame.setSize(220, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Container contentPane = frame.getContentPane();
        contentPane.setLayout(null);

        JButton button = new JButton("Hello World!");
        button.setLocation(30, 30);
        button.setSize(150, 100);
        contentPane.add(button);

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

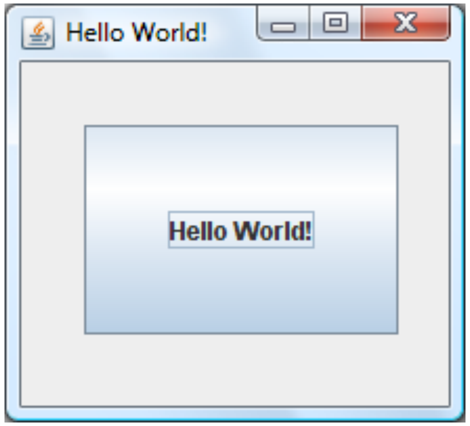


Swing Widgets

THE UNIVERSITY OF AUCKLAND

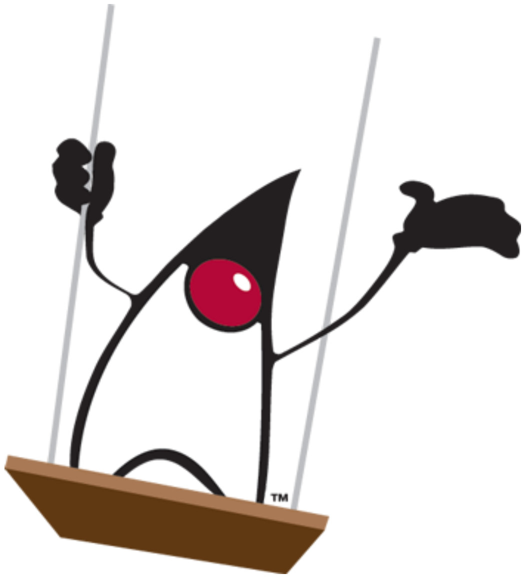
```
JComponent
void setEnabled(boolean)
void setVisible(boolean)
void setX(int)
void setY(int)
void setWidth(int)
void setHeight(int)
void setFont(Font)
void setForeground(Color)
void setBackground(Color)
void paint(Graphics)
...
```

```
Container
Component add(Component)
void remove(Component)
...
```



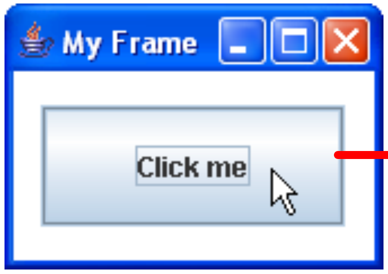
- All widgets inherit from **JComponent**
  - Generic properties for position, size, colors, ...
  - Every widget adds specific properties (e.g. text for button etc.)
- All container widgets inherit from **Container**
  - Generic methods for adding/removing children

Swing Events

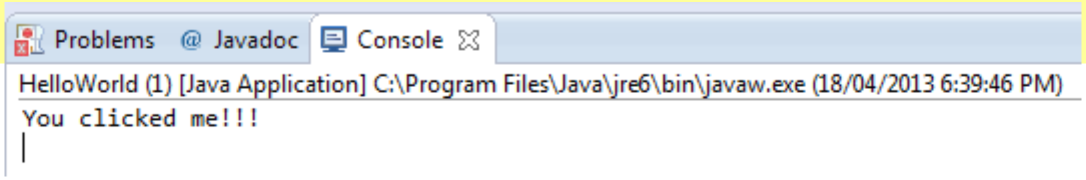


Handling Events

1. When the user does something (click, mouse move, key press, etc) an **event object** is created
2. The event object is sent to the right **target widget** (depending on the mouse position or input focus)
3. Target widget has installed an **event listener** object for the event, using **`add...Listener()`** method
4. The right handler method in the listener object is called, with the event object as argument



```
class MyActionListener implements ActionListener
{
  public void actionPerformed(
          java.awt.event.ActionEvent e) {
    System.out.println("You clicked me!!!");
  }
}
```



```
HelloWorld (1) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (18/04/2013 6:39:46 PM)
You clicked me!!!
```

Swing Hello World with Events

```
...
public class HelloWorld {
   public static void main(String[] args) {
      ...
      JButton button = new JButton("Hello World!");
      button.addActionListener(new MyActionListener());
      ...
   }
}
```

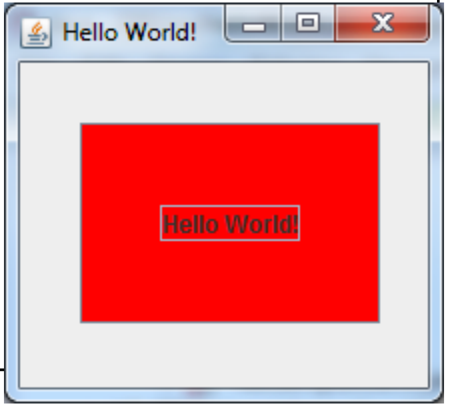
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MyActionListener implements ActionListener {
   public void actionPerformed(ActionEvent e) {
       System.out.println("You clicked me!!!");
      Toolkit.getDefaultToolkit().beep();
   }
}
```

Defining Event Listeners with Anonymous Classes

THE UNIVERSITY OF AUCKLAND

```
...
public class HelloWorld {
   public static void main(String[] args) {
      ...
      final JButton button = new JButton("Hello World!");
      button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            button.setBackround(Color.RED);
        }
      });
      ...
   }
}
```



- Use **`new Classname() {…}`** or **`new Interfacename(){…}`** to create a single object of an anonymous subclass of the given class/interface
- Anonymous classes can access **`final`** variables of their context (i. e. **`final`** variables of the method or class they are created in)

Different Kinds of Swing Events

Low-level events

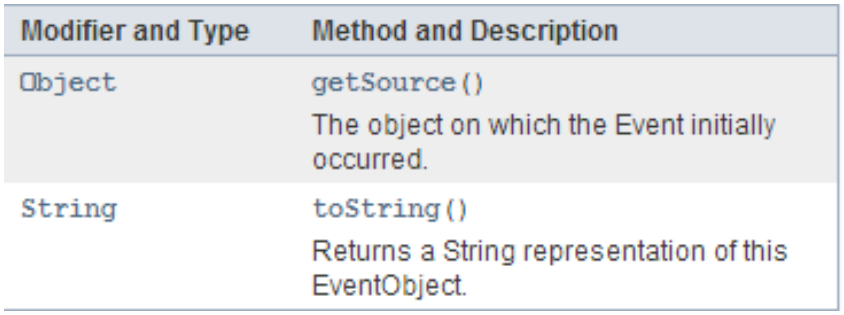
- `MouseEvent`: Component got mouse-down, mouse-move, etc.
- `KeyEvent`: Component got key-press, key-release, etc.
- `ComponentEvent`: Component resized, moved, etc.
- `ContainerEvent`: Container's contents changed because a component was added or removed
- `FocusEvent`: Component got focus or lost focus
- `WindowEvent`: Window opened, closed, etc.

High-level semantic/application events

- `ActionEvent`: Main action of control invoked (e.g. JButton click)
- `AdjustmentEvent`: Value was adjusted (e.g. JScrollBar moved)
- `ItemEvent`: Item was selected or deselected (e.g. in JList)
- `TextEvent`: Text in component has changed
  (e.g in JTextField)

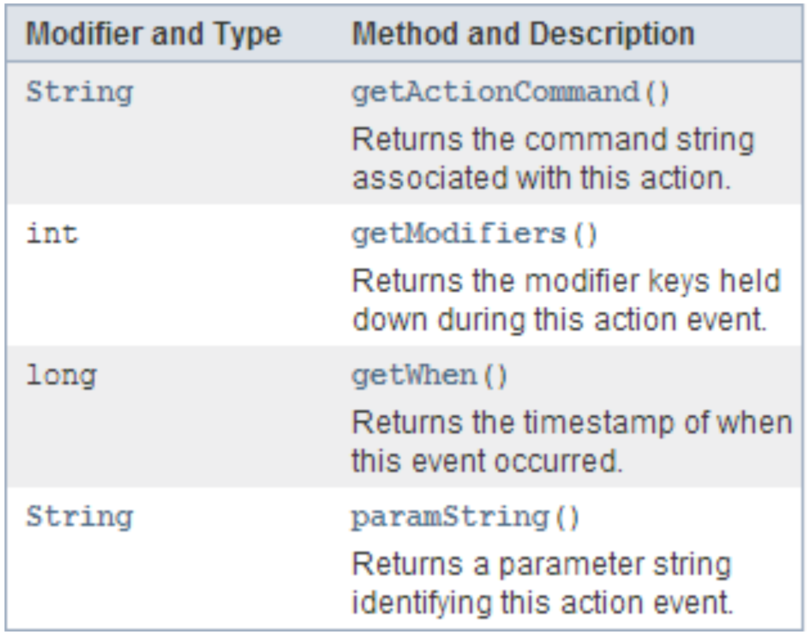
Different Kinds of Swing Events

EventObject



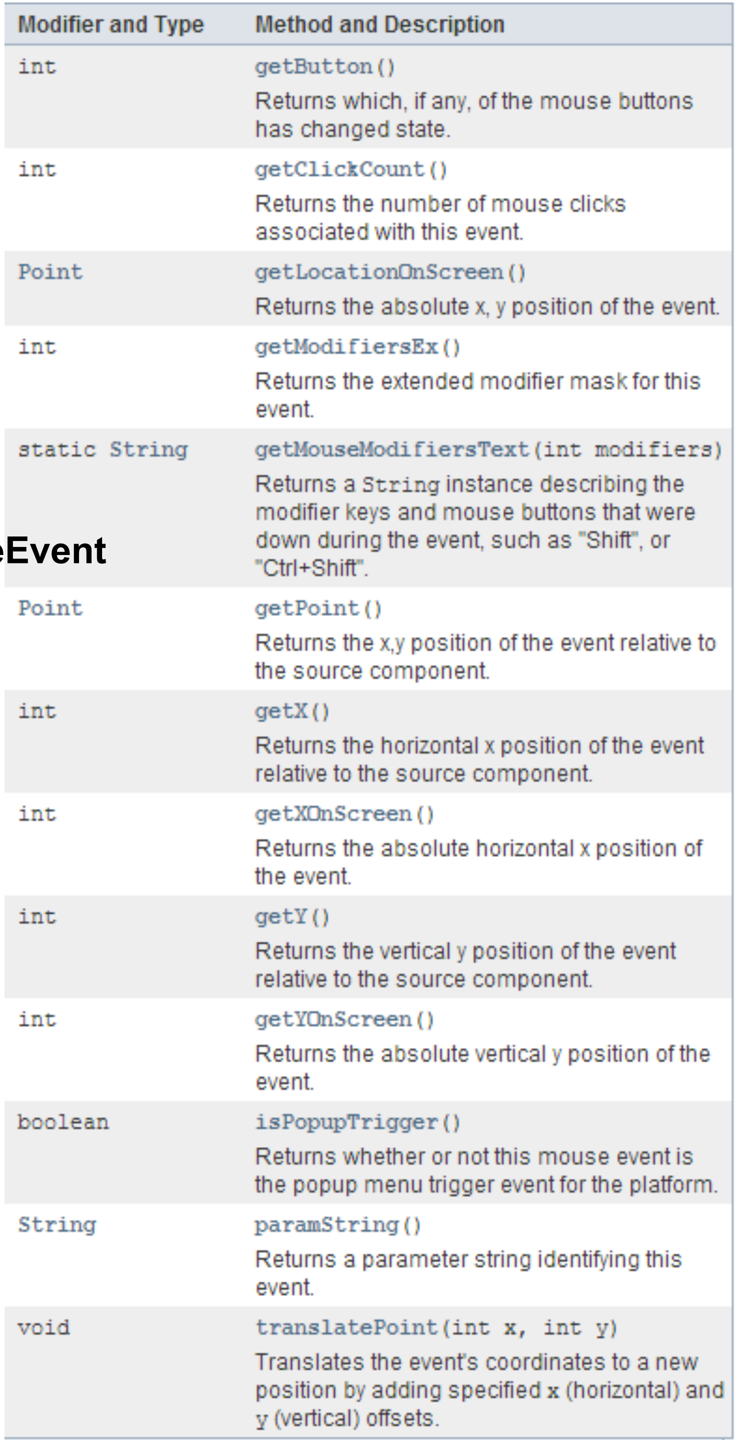
Modifier and Type	Method and Description
`Object`	`getSource()` The object on which the Event initially occurred.
`String`	`toString()` Returns a String representation of this EventObject.

ActionEvent



Modifier and Type	Method and Description
`String`	`getActionCommand()` Returns the command string associated with this action.
`int`	`getModifiers()` Returns the modifier keys held down during this action event.
`long`	`getWhen()` Returns the timestamp of when this event occurred.
`String`	`paramString()` Returns a parameter string identifying this action event.

MouseEvent



Modifier and Type	Method and Description
`int`	`getButton()` Returns which, if any, of the mouse buttons has changed state.
`int`	`getClickCount()` Returns the number of mouse clicks associated with this event.
`Point`	`getLocationOnScreen()` Returns the absolute x, y position of the event.
`int`	`getModifiersEx()` Returns the extended modifier mask for this event.
`static String`	`getMouseModifiersText(int modifiers)` Returns a `String` instance describing the modifier keys and mouse buttons that were down during the event, such as "Shift", or "Ctrl+Shift".
`Point`	`getPoint()` Returns the x,y position of the event relative to the source component.
`int`	`getX()` Returns the horizontal x position of the event relative to the source component.
`int`	`getXOnScreen()` Returns the absolute horizontal x position of the event.
`int`	`getY()` Returns the vertical y position of the event relative to the source component.
`int`	`getYOnScreen()` Returns the absolute vertical y position of the event.
`boolean`	`isPopupTrigger()` Returns whether or not this mouse event is the popup menu trigger event for the platform.
`String`	`paramString()` Returns a parameter string identifying this event.
`void`	`translatePoint(int x, int y)` Translates the event's coordinates to a new position by adding specified `x` (horizontal) and `y` (vertical) offsets.

Events, Listeners, Adapters and Handler Methods

THE UNIVERSITY OF AUCKLAND

Event	Listener / Adapter	Handler Methods
`ActionEvent`	`ActionListener`	`actionPerformed`
`AdjustmentEvent`	`AdjustmentListener`	`adjustmentValueChanged`

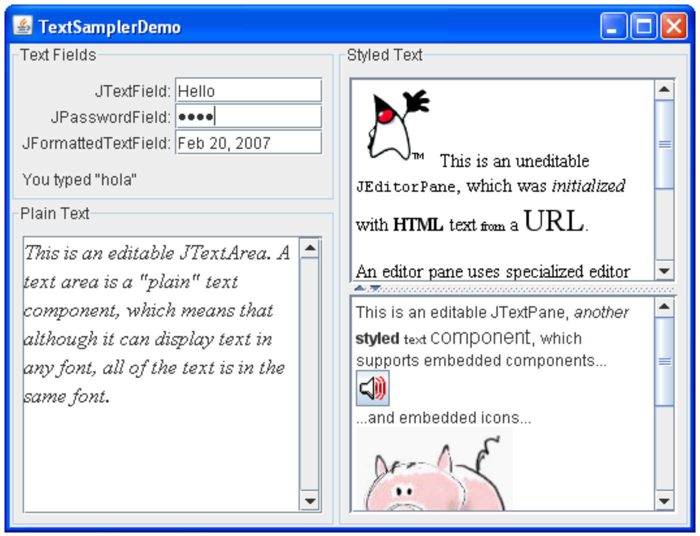
`MouseEvent`	`MouseListener`
	`MouseAdapter`
		`mouseClicked`
		`mouseEntered`
		`mouseExited`
		`mousePressed`
		`mouseReleased`

`KeyEvent`	`KeyListener`
	`KeyAdapter`
		`keyPressed`
		`keyReleased`
		`keyTyped`

`ComponentEvent`	`ComponentListener`
	`ComponentAdapter`
		`componentShown`
		`componentHidden`
		`componentMoved`
		`componentResized`

Adapter classes with empty methods for Listener interfaces with >1 methods

More Widgets



JTextField: Hello
JPasswordField: ••••
JFormattedTextField: Feb 20, 2007
You typed "hola"

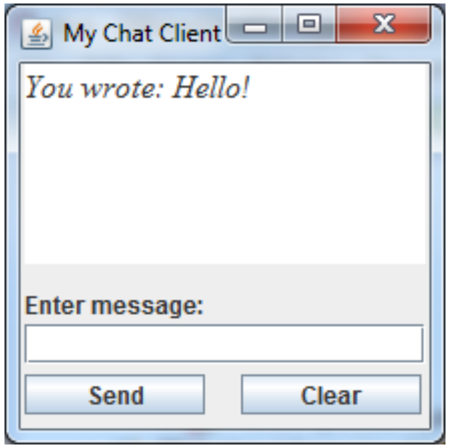
This is an editable JTextArea. A text area is a "plain" text component, which means that although it can display text in any font, all of the text is in the same font.

This is an uneditable `JEditorPane`, which was *initialized* with **HTML** text from a URL.

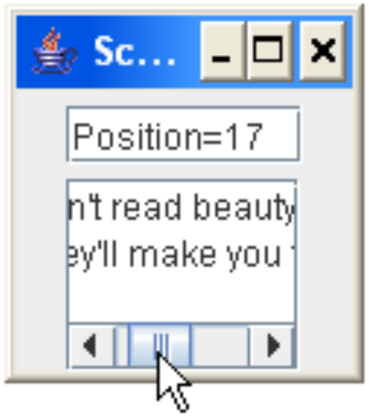
An editor pane uses specialized editor

This is an editable JTextPane, *another* **styled** text component, which supports embedded components...

...and embedded icons...

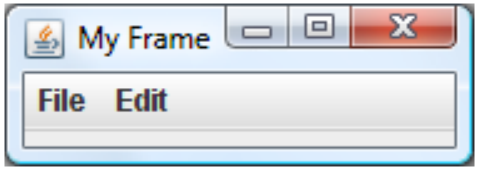


You wrote: Hello!



Position=17

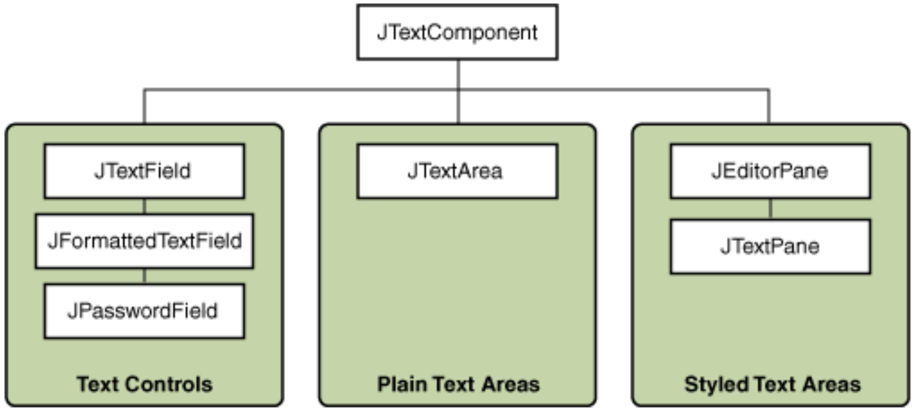
n't read beauty
ey'll make you

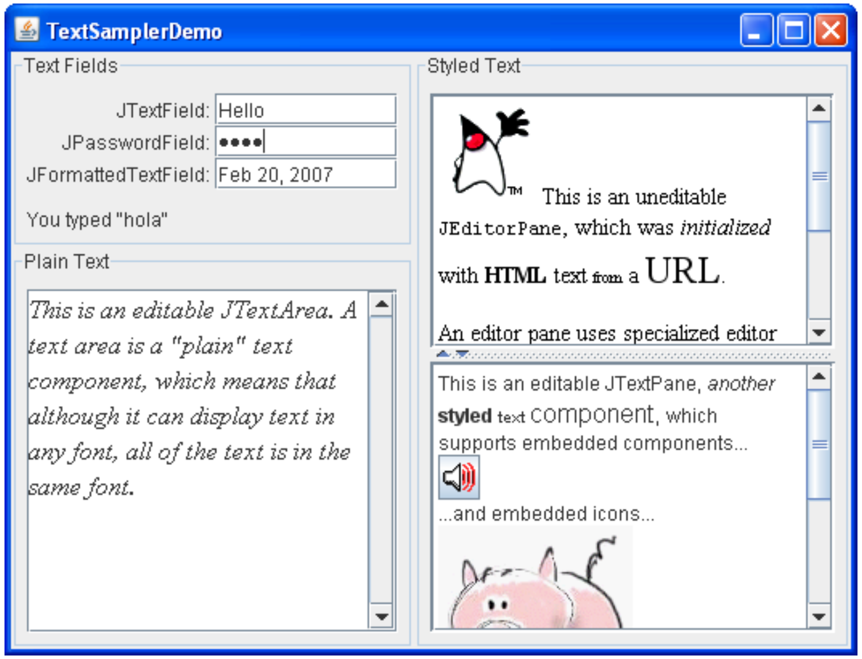


Text Widgets

THE UNIVERSITY OF AUCKLAND

- Often there are several widgets for a certain purpose, to address slightly different requirements
- Usually implemented with **inheritance**:
  - Base functionality in superclass
  - Subclasses with more specific, extended functionality
- Common features of text widgets:
  - Single line vs multi-line
  - Formatting (e.g. for dates)
  - Styling (e.g. different fonts)
  - Multimedia objects (e.g. images, sounds, videos)





Hello

●●●●

Feb 20, 2007

You typed "hola"

This is an editable JTextArea. A text area is a "plain" text component, which means that although it can display text in any font, all of the text is in the same font.

This is an uneditable JEditorPane, which was *initialized* with **HTML** text from a URL.

An editor pane uses specialized editor

This is an editable JTextPane, *another* **styled** text component, which supports embedded components...

...and embedded icons...

Chat Client Example Part 1

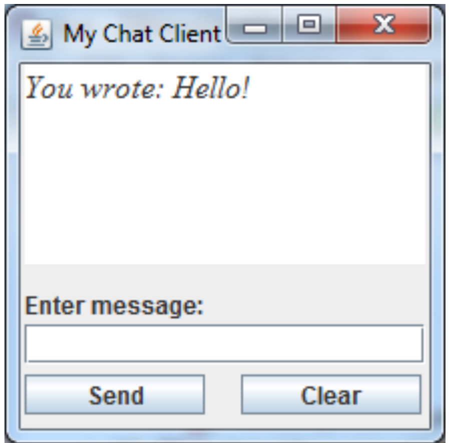
THE UNIVERSITY OF AUCKLAND

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ChatDemo {
  public static void main(String[] args) {
      JFrame frame = new JFrame("My Chat Client");
      frame.setSize(220, 220);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container contentPane = frame.getContentPane();
      contentPane.setLayout(null);

      final JTextArea textArea = new JTextArea();
      textArea.setFont(new Font("Serif", Font.ITALIC, 16));
      textArea.setLineWrap(true);
      textArea.setLocation(2, 0);
      textArea.setSize(200, 100);
      contentPane.add(textArea);

    // to be continued...
```


My Chat Client

You wrote: Hello!

Enter message:

Chat Client Example Part 2

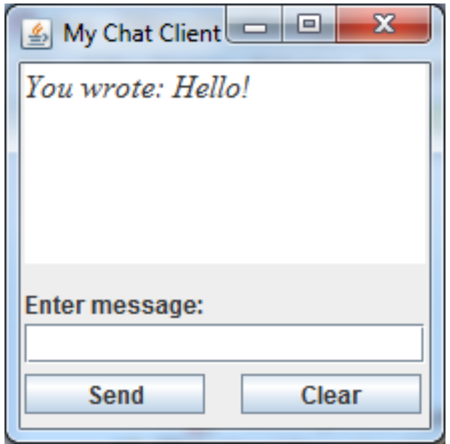
```
JLabel label = new JLabel("Enter message:");
label.setLocation(2, 110);
label.setSize(100, 20);
contentPane.add(label);

final JTextField textField = new JTextField();
textField.setLocation(2, 130);
textField.setSize(200, 20);
contentPane.add(textField);

JButton sendButton = new JButton("Send");
sendButton.setLocation(2, 155);
sendButton.setSize(90, 20);
contentPane.add(sendButton);

JButton clearButton = new JButton("Clear");
clearButton.setLocation(110, 155);
clearButton.setSize(90, 20);
contentPane.add(clearButton);

frame.setVisible(true);
// to be continued...
```



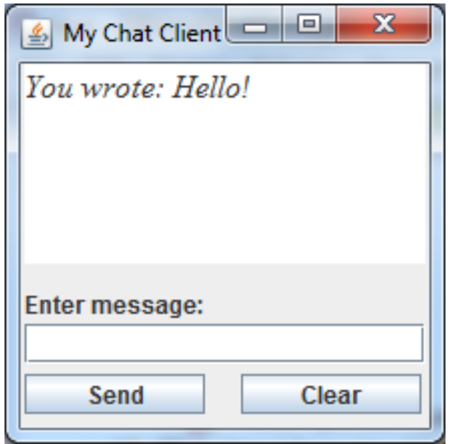
You wrote: Hello!

Chat Client Example Part 3

THE UNIVERSITY OF AUCKLAND

```
    sendButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                textArea.setText(
                textArea.getText()
                + "You wrote: "
                + textField.getText()
                    + "\n");
                textField.setText("");
                textField.requestFocus();
            }
        });

        clearButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                textArea.setText("");
            }
        });
  } // end of main()
} // end of class
```



You wrote: Hello!

Summary

- Swing is a GUI toolkit for Java
  - GUI as containment hierarchy of widgets
  - Event objects and event listeners

- There are various widgets in the Swing API
  - Text widgets can be used to display different kinds of information
  - JScrollPane's can be used to fit a lot of content into a small area
  - Menu bars can be used to organize application functions

References:

http://java.sun.com/docs/books/tutorial/uiswing/

http://www.javabeginner.com/java-swing-tutorial.htm

Quiz

THE UNIVERSITY OF AUCKLAND

1. What is the difference between low-level and high-level events?
2. What kind of information is found in an ActionEvent?
3. What is an anonymous class and why is it useful for specifying event handlers?
4. Write a user interface for the Tic Tac Toe game using Swing.