

CSC 222: Computer Programming II

Spring 2005

Java event handling

- events, event sources, event listeners
- user interaction with buttons: JFrame, JButton
- text display with labels: JLabel
- organizing the interface: JPanel
- user input with text fields: JTextField
- layout management: BorderLayout, GridLayout

1

events & interfaces

using BlueJ, the user was in control of execution

- could create an object, click to call a method, enter parameter values at prompt

in standard Java, the main method automates this interaction

- the statements in the main method specify the sequence of actions to perform

it is possible to create a graphical user interface (GUI) to return user control

- utilize windows, buttons, text boxes, etc.
- the user can control execution by interacting with these elements
- events (window drags, button clicks, mouse moves, ...) trigger actions

when defining an event-driven program, must identify which events are relevant

- e.g., a button doesn't care whether the mouse moves, only if it clicks

2

ActionEvent & ActionListener

as we saw with the Timer class, an `ActionListener` object specifies code that is to be executed when a particular event occurs

- `ActionListener` is an interface defined in
`java.awt.event.ActionListener`
- requires an `actionPerformed` method
- if an object is to be controlled by an event, associate an `ActionListener` with it

```
class Timeout implements ActionListener
{
    public void actionPerformed(ActionEvent event)
    {
        System.out.println("Sorry, time's up.");
        System.exit(0);
    }
}

Timer t = new Timer(TIME_LIMIT, new Timeout());
```

3

Buttons

the simplest graphical interface element is a button

- `JButton` class is defined in `javax.swing.JButton`
- when you create a button, must specify text that appears on it

```
JButton button = new JButton("Click me");
```



the event relevant to a button is being clicked

- associate an `ActionListener` with the button to specify what to do when clicked

```
class ClickListener implements ActionListener
{
    public void actionPerformed(ActionEvent event)
    {
        System.out.println("Thanks, I needed that.");
    }
}

button.addActionListener( new ClickListener() );
```

4

Button example

to be visible & used, a button must be added to a frame (window)

- `JFrame` class defined in `javax.swing.JFrame`
- `setSize` sets the dimensions of the frame
- `setDefaultCloseOperation` allows user to close frame by clicking on X
- `add` puts the button in the frame
- `setVisible` makes the frame visible on the screen

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

public class ButtonTester
{
    public static final int FRAME_WIDTH = 100;
    public static final int FRAME_HEIGHT = 60;

    public static void main(String[] args)
    {
        JButton button = new JButton("Click me");

        class ClickListener implements ActionListener
        {
            public void actionPerformed(ActionEvent event)
            {
                System.out.println("Thanks, I needed that.");
            }
        }
        button.addActionListener( new ClickListener() );

        JFrame frame = new JFrame();
        frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

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

5

Labels & panels

the previous example displays text in console window at a click of the button

- not all that useful – don't want to have to go back and forth from frame to console

using a *label*, the program can write text directly into the frame

- `JLabel` class is defined in `javax.swing.JLabel`
- when you construct a label, specify its initial value

```
JLabel label = new JLabel("I'm waiting...");
```

labels are passive objects – they don't react to events

- but other active elements can change the label text (using `setText`)

```
JButton button = new JButton("Click me");
class ClickListener implements ActionListener
{
    public void actionPerformed(ActionEvent event)
    {
        label.setText("Thanks, I needed that!");
    }
}
button.addActionListener(new ClickListener());
```

6

Label example

only one item can be added to a frame

- if want more than one (e.g., button & label), then group together into a panel
- JPanel class is defined in javax.swing.JPanel
- can add multiple objects to a panel
- then add the panel to the frame

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

public class LabelTester
{
    public static final int FRAME_WIDTH = 200;
    public static final int FRAME_HEIGHT = 200;

    public static void main(String[] args)
    {
        final JLabel label = new JLabel("I'm waiting...");

        JButton button = new JButton("Click me");
        class ClickListener implements ActionListener
        {
            public void actionPerformed(ActionEvent event)
            {
                label.setText("Thanks, I needed that!");
            }
        }
        button.addActionListener(new ClickListener());

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

        JFrame frame = new JFrame();
        frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(panel);
        frame.setVisible(true);
    }
}
```

7

getText / setText

previous example was somewhat pointless

- subsequent clicks produced the same label text (no change)

using the `getText` method, can access the value of a label

- here, access the label text and append to it on each click
- for an inner class to access an object from the method, it must be `final`

```
// IMPORT STATEMENTS OMITTED

public class LabelTester
{
    public static final int FRAME_WIDTH = 200;
    public static final int FRAME_HEIGHT = 200;

    public static void main(String[] args)
    {
        final JLabel label = new JLabel("I'm waiting...");

        JButton button = new JButton("Click me");
        class ClickListener implements ActionListener
        {
            public void actionPerformed(ActionEvent event)
            {
                String text = label.getText();
                if (text.equals("I'm waiting for you..")) {
                    label.setText("Thanks!");
                }
                else {
                    label.setText(text + " Thanks again.");
                }
            }
        }
        button.addActionListener(new ClickListener());

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

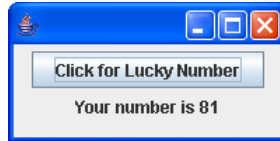
        JFrame frame = new JFrame();
        frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(panel);
        frame.setVisible(true);
    }
}
```

8

Simple application

suppose we wanted to generate a lucky number

- user clicks on a button, lucky number appears below



GUI elements:

- need button to initiate number generation
- need label to display the number
- need panel to contain the button & label, frame to display the panel

program logic:

- need to generate random number – Math.random? Die?

9

Random numbers

Random class is defined
in `java.util.Random`

- `nextInt` method
generates a random integer
in range `[0, parameter)`

again: `randGen` must be
declared `final` to
allow the inner class
to have access

- can call methods to alter
the state of a final object,
but can't reassign it

```
import java.awt.event.ActionEvent;
...
import java.util.Random;

public class LuckyNumber
{
    public static final int FRAME_WIDTH = 200;
    public static final int FRAME_HEIGHT = 100;
    public static final int MAX_NUM = 100;

    public static void main(String[] args)
    {
        final Random randGen = new Random();
        final JLabel label = new JLabel("");

        JButton button = new JButton("Click for Lucky Number");
        class ClickListener implements ActionListener
        {
            public void actionPerformed(ActionEvent event)
            {
                label.setText("Your number is "
                    + (randGen.nextInt(MAX_NUM)+1));
            }
        }
        button.addActionListener(new ClickListener());

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

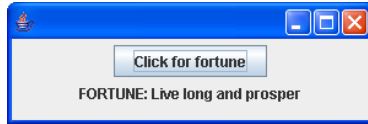
        JFrame frame = new JFrame();
        frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(panel);
        frame.setVisible(true);
    }
}
```

10

More interesting variant

suppose we wanted to generalize the lucky number program

- instead of generating a lucky number, want to generate a fortune



GUI elements:

- need button, label, panel & frame (as before)

program logic:

- need to store a list of possible fortunes – array of Strings? ArrayList of Strings?
- need to generate random index, select fortune at that index

11

Random fortunes

arrays are handy if you know their contents ahead of time

- can list explicit contents when declaring/creating

```
// IMPORT STATEMENTS OMITTED
public class FortuneCookie
{
    public static final int FRAME_WIDTH = 300;
    public static final int FRAME_HEIGHT = 100;
    public static final String[] FORTUNES =
        {"Live long and prosper",
         "An apple a day keeps the doctor away.",
         "Don't take any wooden nickels."};

    public static void main(String[] args)
    {
        final Random randGen = new Random();
        final JLabel fortuneLabel = new JLabel("");

        JButton button = new JButton("Click for fortune");
        class ClickListener implements ActionListener
        {
            public void actionPerformed(ActionEvent event)
            {
                fortuneLabel.setText("FORTUNE: "
                    + FORTUNES[randGen.nextInt(FORTUNES.length)]);
            }
        }
        button.addActionListener(new ClickListener());

        JPanel panel = new JPanel();
        panel.add(button);
        panel.add(fortuneLabel);

        JFrame frame = new JFrame();
        frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(panel);
        frame.setVisible(true);
    }
}
```

12

Click counts

suppose we wanted to keep a count of the occurrences of some event

- e.g., display count of number of times mouse has been clicked

ATTEMPT: define a counter, have the ActionListener increment & display

```
public static void main(String[] args)
{
    final JLabel textLabel = new JLabel("Number of clicks = 0");

    int clickCount = 0;

    JButton button = new JButton("Click Me");
    class ClickListener implements ActionListener
    {
        public void actionPerformed(ActionEvent event)
        {
            clickCount++;
            textLabel.setText("Number of clicks = " + clickCount);
        }
    }
    button.addActionListener(new ClickListener());

    . . .
}
```

PROBLEM?

won't work –
inner class can
only access final
values

13

Click count

VERSION 1: could define a Counter class that encapsulates a counter

- could define a final Counter object, call increment method in ActionListener

VERSION 2: could add state to the ActionListener class

- maintain the counter as a field, can then increment & display the field

```
public static void main(String[] args)
{
    final JLabel textLabel = new JLabel("Number of clicks = 0");

    JButton button = new JButton("Click Me");
    class ClickListener implements ActionListener
    {
        private int clickCount;
        public ClickListener()
        {
            clickCount = 0;
        }
        public void actionPerformed(ActionEvent event)
        {
            clickCount++;
            textLabel.setText("Number of clicks = " + clickCount);
        }
    }
    button.addActionListener(new ClickListener());

    . . .
}
```

14

VERSION 3

can access a
number label as text

convert to a number
using the
`Integer.parseInt`
method

increment and put
the number back in
the label as text



```
// IMPORT STATEMENTS OMITTED

public class ClickCounter
{
    public static final int FRAME_WIDTH = 200;
    public static final int FRAME_HEIGHT = 100;

    public static void main(String[] args)
    {
        final JLabel textLabel = new JLabel("Number of clicks = ");
        final JLabel countLabel = new JLabel("0");

        JButton button = new JButton("Click Me");
        class ClickListener implements ActionListener
        {
            public void actionPerformed(ActionEvent event)
            {
                int newCount = Integer.parseInt(countLabel.getText())+1;
                countLabel.setText(""+newCount);
            }
        }
        button.addActionListener(new ClickListener());

        JPanel panel = new JPanel();
        panel.add(button);
        panel.add(textLabel);
        panel.add(countLabel);

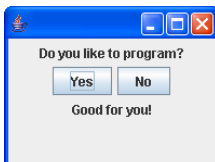
        JFrame frame = new JFrame();
        frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(panel);
        frame.setVisible(true);
    }
}
```

15

Multiple buttons

if multiple buttons
appear in a frame, it
is common for each
to define its own
`ActionListener`

if actions to be
performed are
similar, can have a
single class with
fields & constructor
parameters



```
// IMPORT STATEMENTS OMITTED

public class Quizzer
{
    public static final int FRAME_WIDTH = 200;
    public static final int FRAME_HEIGHT = 150;

    public static void main(String[] args)
    {
        final Random randGen = new Random();

        final JLabel questionLabel = new JLabel("Do you like to program?");
        final JLabel answerLabel = new JLabel("");

        class ClickListener implements ActionListener
        {
            private String labelMessage;

            public ClickListener(String msg)
            {
                labelMessage = msg;
            }

            public void actionPerformed(ActionEvent event)
            {
                answerLabel.setText(labelMessage);
            }
        }

        JButton yesButton = new JButton("Yes");
        yesButton.addActionListener(new ClickListener("Good for you!"));

        JButton noButton = new JButton("No");
        noButton.addActionListener(new ClickListener("Give it some time."));

        . . .
    }
}
```

16

text fields & areas

Basic GUI elements:

- labels → output in a frame
- buttons → user control (i.e., initiating an event)
- text fields/areas → user input

a text field is a single-line box in which the user can enter text

- `JTextField` class is defined in `javax.swing.JTextField`

```
JTextField nameField = new JTextField(FIELD_WIDTH);
```

- similar to `JLabel`, can access contents using `getText`,

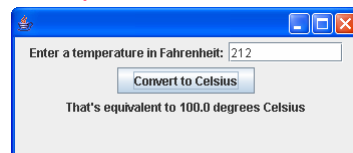
```
String userName = nameField.getText();  
textLabel.setText("Hi " + userName + ", nice to meet you.");
```

- can even update the contents of a text field using `setText`
but, usually, a label looks better for output

17

```
// IMPORT STATEMENTS OMITTED  
public class TempConverter  
{  
    public static final int FRAME_WIDTH = 350;  
    public static final int FRAME_HEIGHT = 150;  
    public static final int FIELD_WIDTH = 10;  
  
    public static double fahrToCelsius(double fahrTemp)  
    {  
        return (5.0/9.0) * (fahrTemp - 32);  
    }  
  
    public static void main(String[] args)  
    {  
        final JLabel inputLabel = new JLabel("Enter a temperature in Fahrenheit:");  
        final JLabel outputLabel = new JLabel("");  
        final JTextField tempField = new JTextField(FIELD_WIDTH);  
  
        class ClickListener implements ActionListener  
        {  
            public void actionPerformed(ActionEvent event)  
            {  
                double celsius = fahrToCelsius(Double.parseDouble(tempField.getText()));  
                outputLabel.setText("That's equivalent to " + celsius + " degrees Celsius");  
            }  
        }  
  
        JButton convertButton = new JButton("Convert to Celsius");  
        convertButton.addActionListener(new ClickListener());  
  
        JPanel panel = new JPanel();  
        panel.add(inputLabel);  
        panel.add(tempField);  
        panel.add(convertButton);  
        panel.add(outputLabel);  
  
        JFrame frame = new JFrame();  
        frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        frame.add(panel);  
        frame.setVisible(true);  
    }  
}
```

Temperature converter



18

In-class exercise

suppose we wanted to allow for temperature conversion either way

- user can enter temperature in Fahrenheit, convert to Celsius
- user can enter temperature in Celsius, convert to Fahrenheit

what should the frame look like?

additional GUI elements?

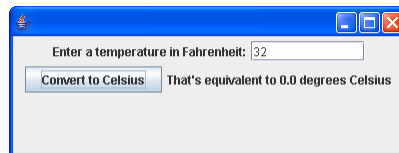
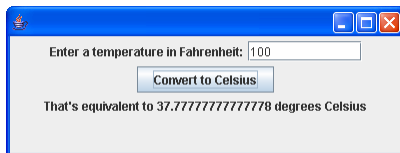
associated functionality?

19

Layout management

so far, we have had limited control over the layout of GUI elements

- by default, a panel uses *flow layout* (elements are left-to-right, wrap as needed)
- sometimes this is sufficient, but sometimes not

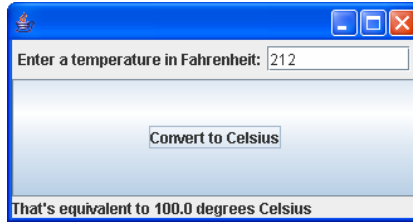


the `java.awt` has layout managers that can be used to place elements

- `BorderLayout`: specify NORTH, EAST, CENTER, WEST, or SOUTH when add to panel
- `GridLayout`: can specify number of rows & columns, then add in order
- can only place one element in a panel position – if more than one (e.g., label + field), then nest inside an inner panel

20

Temperature layout



```
import java.awt.BorderLayout;  
  
...  
  
JPanel innerPanel = new JPanel();  
innerPanel.add(inputLabel);  
innerPanel.add(tempField);  
  
JPanel panel = new JPanel();  
panel.setLayout(new BorderLayout());  
panel.add(innerPanel, BorderLayout.NORTH);  
panel.add(convertButton, BorderLayout.CENTER);  
panel.add(outputLabel, BorderLayout.SOUTH);  
  
...
```

```
import java.awt.GridLayout  
  
...  
  
JPanel innerPanel = new JPanel();  
innerPanel.add(inputLabel);  
innerPanel.add(tempField);  
  
JPanel panel = new JPanel();  
panel.setLayout(new GridLayout(3, 1));  
panel.add(innerPanel);  
panel.add(convertButton);  
panel.add(outputLabel);  
  
...
```

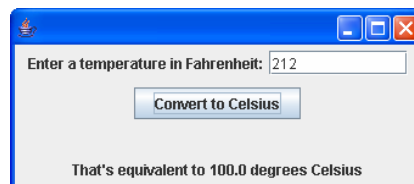
21

Temperature layout

```
import java.awt.BorderLayout;  
  
...  
  
JPanel innerPanel = new JPanel();  
innerPanel.add(inputLabel);  
innerPanel.add(tempField);  
  
JPanel buttonPanel = new JPanel();  
buttonPanel.add(convertButton);  
  
JPanel outputPanel = new JPanel();  
outputPanel.add(outputLabel);  
  
JPanel panel = new JPanel();  
panel.setLayout(new BorderLayout());  
panel.add(innerPanel, BorderLayout.NORTH);  
panel.add(buttonPanel, BorderLayout.CENTER);  
panel.add(outputPanel, BorderLayout.SOUTH);  
  
...
```

even when there is only one element in a position, can be useful to place in an inner panel

- keeps the button from filling the space
- centers the element in the space



22

In-class exercise

use a layout manager to control the layout of your temperature conversion page

BorderLayout or GridLayout?

nested panels?

23

Text area

a text area is similar to a text field except it has multiple rows

- JTextArea class is defined in javax.swing.JTextArea

```
JTextArea textArea = new JTextArea(NUM_ROWS, NUM_COLS);
```

- can access the contents of a text area using getText
- can assign a value using setText, or append text using append

```
textArea.setText("Howdy\n");  
textArea.append("Do");
```

- can designate a text area to be for output only, or set line wrapping

```
textArea.setEditable(false);  
textArea.setLineWrap(true);
```

- can add a scroll bar using the JScrollPane class

```
JScrollPane scrollPane = new JScrollPane(textArea);
```

24

Hailstone sequence

```
public class Hailstone
{
    public static final int FRAME_WIDTH = 350;
    public static final int FRAME_HEIGHT = 450;
    public static final int FIELD_WIDTH = 6;
    public static final int AREA_WIDTH = 20;
    public static final int AREA_HEIGHT = 30;

    public static void main(String[] args)
    {
        final JLabel inputLabel = new JLabel("Enter a starting value:");

        final JTextField startField = new JTextField(FIELD_WIDTH);

        final JTextArea outputArea = new JTextArea(AREA_WIDTH, AREA_HEIGHT);
        outputArea.setEditable(false);
        outputArea.setLineWrap(true);
        JScrollPane outputPane = new JScrollPane(outputArea);

        class ClickListener implements ActionListener
        {
            public void actionPerformed(ActionEvent event)
            {
                int num = Integer.parseInt(startField.getText());
                outputArea.setText("" + num + "\n");

                while (num != 1) {
                    if (num % 2 == 0) {
                        num = num / 2;
                    }
                    else {
                        num = 3*num + 1;
                    }
                    outputArea.append("" + num + "\n");
                }
            }
        }
    }
}
. . .
```

