Assignment 5

GUI Interface for IntelligentTicTacToe

Presented in class 4-5-2013, with a few minor corrections.
Key Concepts of Swing
(as relevant to this project)

- Always start with a JFrame.
- Add contents to the main container, called content pane.
- Elements can be widgets, or containers (JPanels for this project) to hold more elements.
- Every container has a layout, which sets the size and position of widgets.
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;
import java.awt.event.*;

public class test extends JFrame{
    public test(){
        pack();
    }
    public static void main(String[] args){
        test t = new test();
        t.setVisible(true);
    }
}
public test(){
    setDefaultCloseOperation(
        JFrame.EXIT_ON_CLOSE);
    setTitle("My Frame");
    pack();
}

Note, has to be resized manually.
Common Widgets

- JLabel: a non-editable block of text in the frame
- JTextField: a single-line block for text
- JButton: something to click on
Layout Managers

- Layout managers are the preferred way to arrange widgets on the screen.
- Absolute positioning is strongly deterred.
- Layout managers allow widget to dynamically move as the screen (or other widgets) change size.
  - Also allow automatic sizing of screen using the pack method.
public test(){
    Container cp = getContentPane();
    cp.setLayout(new FlowLayout());
    cp.add(new JButton("This is a button"));
    cp.add(new JLabel("This is a label"));
    cp.add(new JTextField("This is a text field"));
    pack();
}

- Simplest layout. Widgets flow from right to left - wrap to next line as needed.
- Widgets automatically get their preferred size (usually looks good)
public test(){
    Container cp = getContentPane();
    cp.setLayout(new GridLayout(2,2,10,10));
    cp.add(new JButton("ONE"));
    cp.add(new JButton("TWO"));
    cp.add(new JButton("THREE"));
    cp.add(new JButton("FOUR"));
    pack();
}

- Widgets arranged in grid. Easy to understand.
- Allows horizontal/vertical gaps (ex. 10,10 above)
- Widgets automatically get expanded to fill container.
- Rarely used for arranging widgets, unless nested inside a JPanel using a non-expanding layout (ie. rightInnerPanel on slide 13).
  - More often used for arranging nested JPanels.
BorderLayout

public test(){
    Container cp = getContentPane();
    cp.setLayout(new BorderLayout());
    cp.add(new JButton("NORTH"), BorderLayout.NORTH);
    cp.add(new JButton("SOUTH"), BorderLayout.SOUTH);
    cp.add(new JButton("EAST"), BorderLayout.EAST);
    cp.add(new JButton("WEST"), BorderLayout.WEST);
    cp.add(new JButton("CENTER"), BorderLayout.CENTER);
    pack();
}

- Divides container into regions: north, south, east west, center
- Only one component per region.
- Center component expands to fill extra space (what the user usually expects)
  - Very handy. Matches common usage pattern.
GridBagLayout

- Commonly referenced in forum posts you may find using Google.
- This layout is very powerful, and very complex. You do not need to use it for this project.
Misc. Comments about Layout Managers

- Most of the time invested in developing GUIs is spent trying to get the layout managers to cooperate with what you want to do.
  - How many nested panels? What layouts to use? How to get gaps?
- Very few people can develop layout manager code without consulting online reference material.
- Use Google!
- Also, Visual Guide to Layout Managers:
  http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html
Components of the GUI

- **rightPanel (JPanel)**
  - `FlowLayout()`

- **boardButtons**
  - `JButton[]`

- **ContentPane (Container)**
  - `BorderLayout`

- **boardPanel (JPanel)**
  - `GridLayout(n,n,20,20)`

- **bottomPanel (JPanel)**
  - `FlowLayout(CENTER)`

- **rightInnerPanel (JPanel)**
  - `GridLayout(4,2,10,10)`

- **left column**: unnamed `JLabel`
  - (since we don't need to edit them)

- **ContentPane (JLabel)**
  - `size (JTextField)`
  - `player (JLabel)`
  - `move (JLabel)`
  - `winner (JLabel)`

- **bottomPanel (JButton)**
  - `newButton (JButton)`
  - `adviseButton (JButton)`
  - `quitButton (JButton)`
Event Handling

- How do we respond to user input?
- General idea is to register an object as a listener. Methods of the listener get invoked automatically when the user does something.
- When a user clicks a JButton, java calls that an action. An `ActionEvent` is created and passed to the `actionPerformed` method of an `ActionListener`.
- A common approach for beginners is to use the GUI as the `ActionListener`.
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;
import java.awt.event.*;

public class EventDemo extends JFrame implements ActionListener{
    private JButton one, two;
    public EventDemo(){
        Container cp = getContentPane();
        cp.setLayout(new GridLayout(2,1,10,10));
        cp.add(one = new JButton("ONE"));
        cp.add(two = new JButton("TWO"));
        one.addActionListener(this);
        two.addActionListener(this);
        pack();
    }
    public void actionPerformed(ActionEvent e){
        Object source = e.getSource();
        if(source == one)
            System.out.println("User clicked ONE");
        if(source == two)
            System.out.println("User clicked TWO");
    }
    public static void main(String[] args){
        EventDemo ed = new EventDemo();
        ed.setVisible(true);
    }
}
Number Chooser GUI

- `numberChooserGUI`:
  - `buttonPanel` (JPanel) with GridLayout(n,n,20,20)
  - `sizePanel` (JPanel) with FlowLayout()
  - `bottomPanel` (JPanel) with FlowLayout()

- Elements:
  - `sizeField` (JTextField)
  - `rebuildButton` (JButton)
  - `output` (JLabel)
  - Anonymous JLabel
  - Buttons JButton[]

- Layout:
  - Top section: `Number Chooser` with `Button grid size?` and `Rebuild` button
  - Middle section: 16 buttons arranged in a 4x4 grid
  - Bottom section: `Your favorite number is 86`