* AWT(abstract window toolkit)
  + original Java UI framework
  + normalized interface to native OS UI toolkits
  + packages start with java.awt
* Swing
  + pure Java
  + more sophisticated components
  + more easily extended and customized
  + packages tart with javax.swing

Ex:

JFrame main\_frame = new JFrame();

main\_frame.setTitle("Hello World");

main\_frame.setDefaultCloserOperation(Jframe.EXIT\_ON\_CLOSE);

-this creates a window that will appear on your desktop(won't appear yet)

JPanel main\_panel = new JPanel();

main\_frame.setContentPane(main\_panel);

-JPanel = content of the frame

-this JPanel replaces the default content pane

main\_panel.setLayout(new BorderLayout());

-each panel(or container) is associated with a layout manager

-different layout manager provides different types and styles of arrangement, differing in flexibility and sophistication

-the Border layout allows 5 components to be placed in one of 5 areas: NORTH, SOUTH, EAST, WEST, CENTER

-center area is given any extra space

-unfilled areas are collapsed

JLabel hello\_world\_label = new JLabel("Hello World!");

hello\_world\_label.setHorizontalAlignment(SwingConstants.CENTER);

hello\_world\_label.setForeground(Color BLUE);

hello\_world\_label.setBackground(Color.YELLOW);

-JLabel is a simple text component

-property setters for alignment, color, etc

-colors in Java are represented by class java.awt.Color

-Colors are class constants

-SwintConstants = number of constants and enumerations used as symbolic names understood by various methods

main\_panel.add(hello\_world\_label, BorderLaout.CENTER);

main\_frame.pack();

main\_frame.setVisible(true);

-pack() resolves layout geometry

-make the main\_frame visible

* Top level windows
  + JDialog
    - JOptionPane provides quick and easy dialog boxes
      * showConfirmDialog: allows user to confirm a choice with a Yes/No/Cancel response
      * showInputDialog: prompts for text input
      * shwoMessageDialog:shows message, waits for user to acknowledge
    - all of these are model, meaning that the flow of execution is halted until the dialog is dealt with
  + JFrame
    - non-modal top level window
    - may or may not have a window bar
    - operation defined by what is placed in it(OS specific)
    - General pattern:
      * main method of the program sets up UI ina JFrame
      * JFrame made visible
      * rest of program now responds asynchronously to user interface events
  + JApplet
* Layout Managers
  + Border Layout
    - 5 areas: PAGE\_START, LINE\_START, LINE\_END, PAGE\_END
  + Box Layout
    - stacked either horizontally or vertically
  + gridlayout
    - equal sized, regular grid
* Swing Components
  + Text
    - JLabel, JTextField, JTextArea
  + Buttons
    - JButton, JCheckBox, JRadioButton, JToggleButton
  + Sliders
    - JSlider
  + Tons of others

* panel methods:
  + add(element, where) or .add(element);
  + .setLayout(new GridLayout(1,3));
* button methods
  + new JButton("Say Bye");
  + .setActionCommand("hello");
  + .addActionListener(responder);

* UI events
  + UI elements respond to interaction by generation "events"
    - -listeners are registered to receive notification when a particular event occurs
    - different listener interfaces are defined for different events
    - listening method receives event information as a parameter
  + should recognize this as observer/observable
    - UI elements are observable
    - listeners are the observers
  + anything can be a event listener as long as it abides by the interface
    - ActionListener Interface

void actionPerformed(ActionEvent e){...}

* to add listeners to events, do:

addActionListener(ActionListener I){...}

* the ActionListener has a target Element(such as a JLabel) which it updates depending on the events it is listening to

Overview

* Create a top level window
* create a JPanel to be content pane
* fill JPanel with other components
* Connect UI events with "listeners"
* Make top level window visible

Hierarchy

* Component -> Container -> JComponent

Different Listener types as well

* ComponentListener
* FocusListener
* KeyListener
* MouselIstener
* MouseMotionListener
* MouseWheelListener

awt.Container

* ContainerListener

Abstract Button

* ActionListener
* ChangeListener
* ItemListener

Keyboard Concepts

* Keyboard events go to component that currently has "focus"
  + setFocusable(boolean status)
  + isFocusable()
  + requestFocusInWindow()

KeyListener interface

* keyTyped(KeyEvent e)
* keyPressed(KeyEvent e)
* keyReleased(KeyEvent e)

* Key Event methods
  + char getKeyChar()
    - only valid for key typed events
  + int getKeyCode()
  + isAltDown(), isControlDown(), isShiftDown(), isMetaDown()

* Various Mouse Listener methods as well that detect mouse clicks, entering, and releases
* MouseMotionListener
  + mouseDragged(MouseEvent e)
  + mouseMoved(MouseEvent e)
* MouseEvent
  + position info
    - relative to component: getX(), getY(), getPoint()
    - absolute position: getXOnScreen(), getYOnScreen()
  + click info
    - getClickCount()
    - getButton()
  + modifiers
    - isAltDown()
    - isShiftDown()
* Composing Widgets
  + a set of related UI elements that act as a unit within your UI
  + basic idea:
    - Subclass Panel, implement ChangeListener
    - in constructor, create and arrange UI components, addChangeListeners
    - provide methods for attaching listeners, requesting current state of UI, etic
      * void stateChanged(ChangeEvent e){...}
        + inside this method, you should update a label's text or something based on a ChangeEvent
    - could be straight delegation or could be mediated by more sophisticated logic