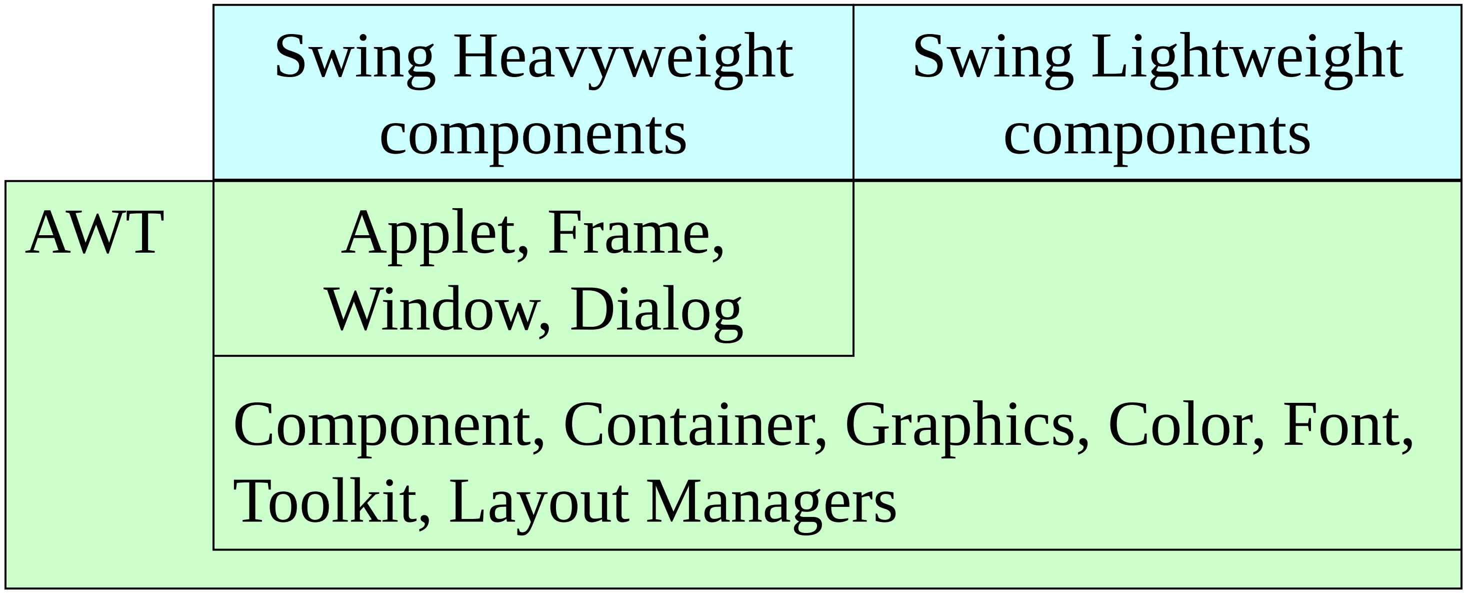
Java GUI & 2D Graphics

Java Foundation Classes: **Swing & AWT, Java 2D API**
Swing is not a replacement for AWT (Abstract Window Toolkit)
Swing classes extend AWT

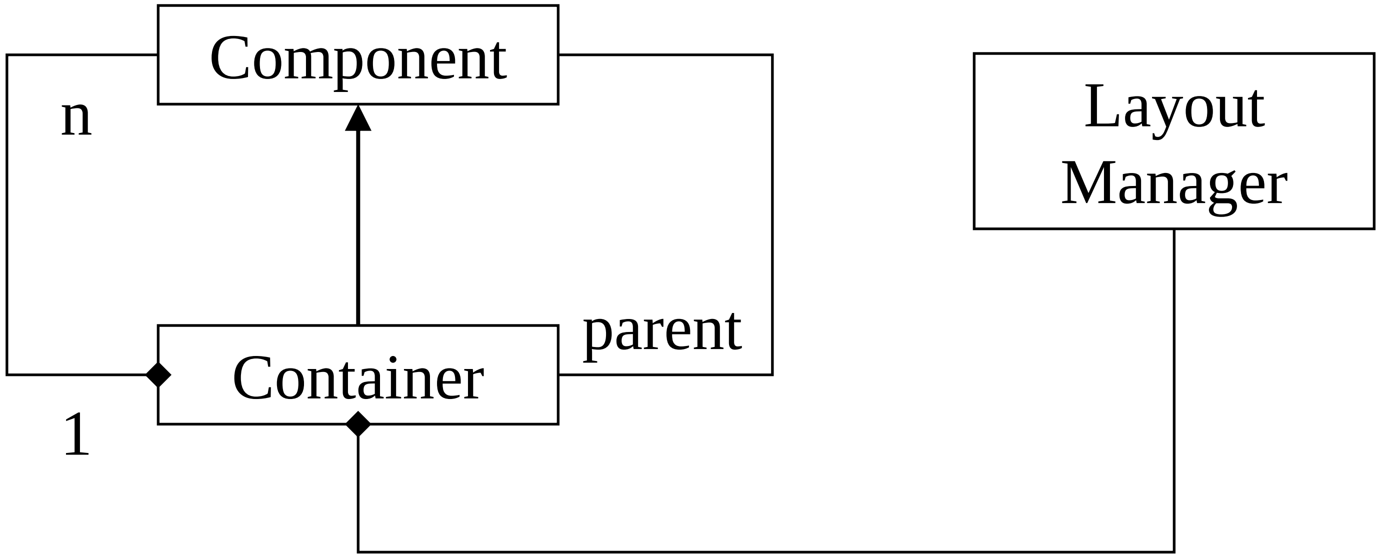

Swing Heavyweight components
Swing Lightweight components
AWT
Applet, Frame, Window, Dialog
Component, Container, Graphics, Color, Font, Toolkit, Layout Managers

Heavyweight components are opaque, windows, drawn by native window system, not as portable.

Lightweight components can have transparent backgrounds, drawn by JVM, portable.

Containers

Java UIs consist of containers using layout managers to control the geometry of UI components.



Java UI components can register listeners (or adapters) to react to events that are "fired off" by components.

components have add*Listener() methods for expected events.

listeners are java interfaces.

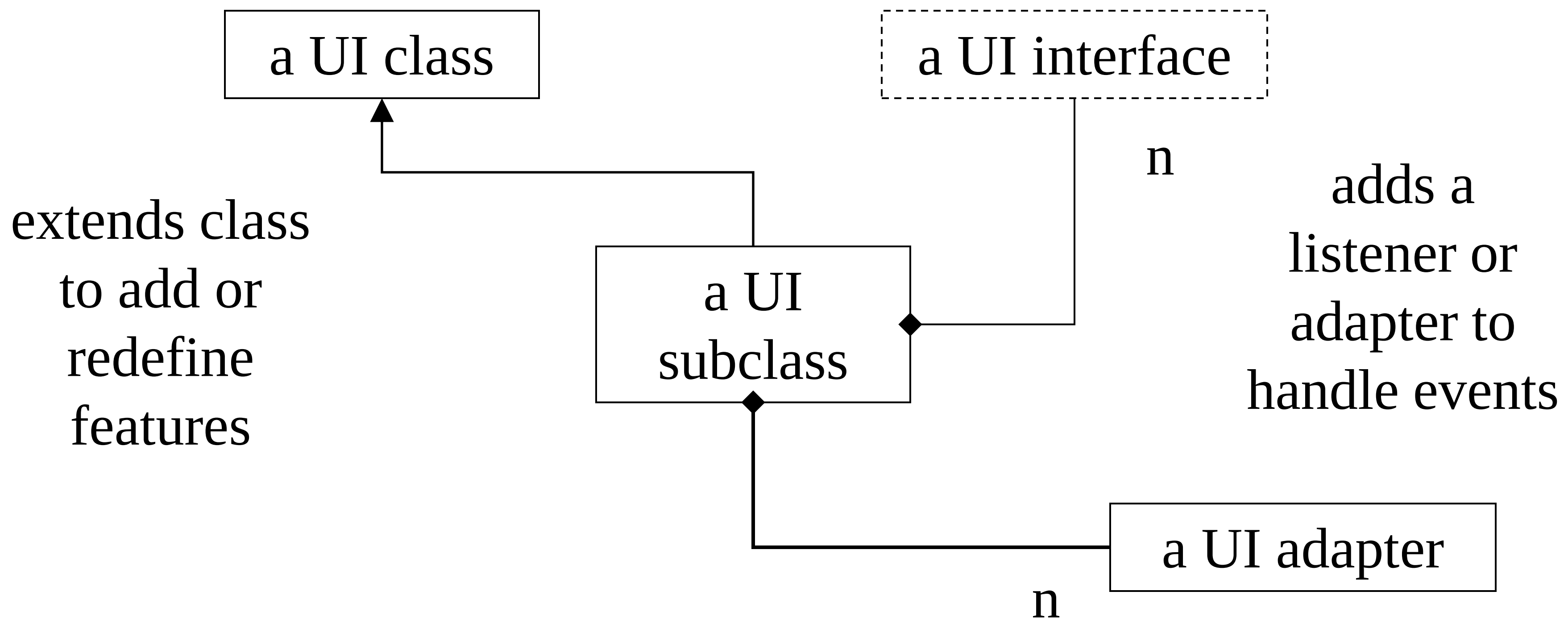
Swing ADTs

classes containers, components, layout managers, events, adapters
JFame, JLabel, BorderLayout, MouseAdapter

interfaces event listeners
ActionListener, MouseListener
interfaces are messy w 2+ methods.
  every implementor must define all methods

adapters convenience classes
Swing provides an adapter class for every listener
  interfaces w/ 2+ methods.
Adapter extends an interface w/ methods w/ null behavior.
  MouseAdapter, WindowAdapter

Java has single inheritance for classes but can implement multiple interfaces.


a UI class
a UI interface
n
adds a listener or adapter to handle events
extends class to add or redefine features
a UI subclass
a UI adapter
n

Inner Classes

Inner class's definition is nested w/in outer class

Named or unnamed inner classes

Scope rules allow reference of outer class variables

Listeners

The listener's interface member functions are invoked when an event is received.

ActionListener, AdjustmentListener, ContainerListener, FocusListener, ItemListener, KeyListener, MouseListener, MouseMotionListener, TextListener, WindowListener.

Events are represented by event classes.

MouseEvent has getX(), getY(), getPoint(), getClickCount(), translatePoint(int x, int y), isPopupTrigger() interfaces.

Not all events have a specific class -- some event classes represent related events and use integer constants to identify the actual event.

WindowEvent handles: activating, deactivating, closing, opening, iconifying, and de-iconifying windows with getWindow() interface

Events

Event handling wiring *"problem"*

Designer of class wants to support an "application specific" behavior when a user initiates an event (request for behavior), but doesn't know what the event will be.

Provides a typed *"callback slot"* reference and set methods to "add" or "remove" an event handler object.

Application developer wants to use or extend existing GUI classes and be able to add "application specific" behaviors (methods) to be performed when user requests them via the GUI control.

Must provide correct type *"callback object"* when setting event handler.

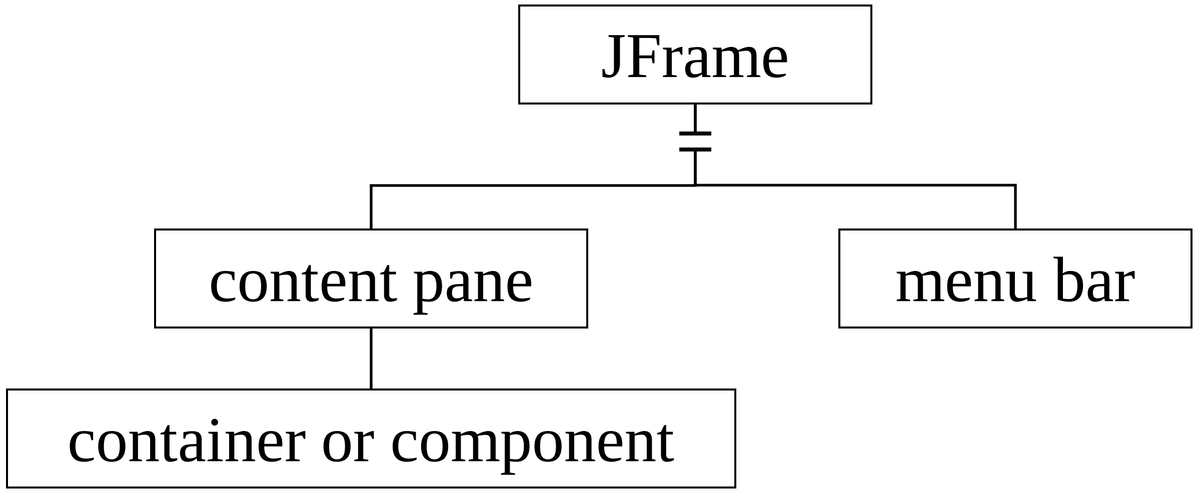
Top Level Containers

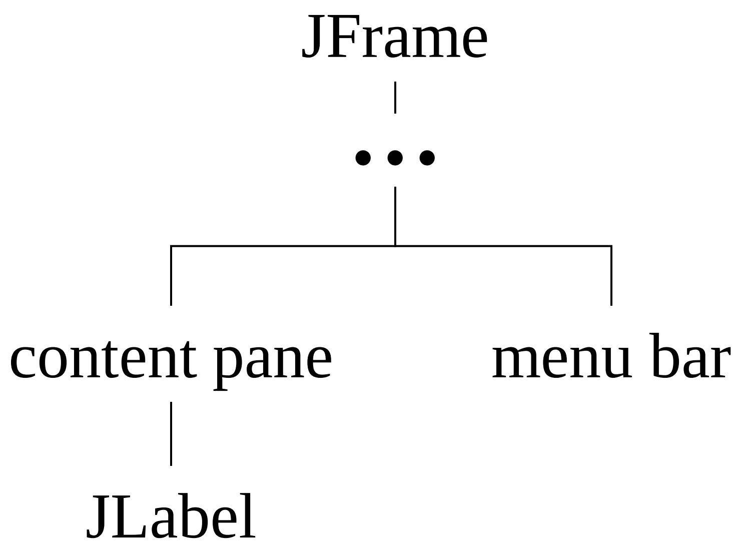
JApplet, JFrame, JDialog, JWindow

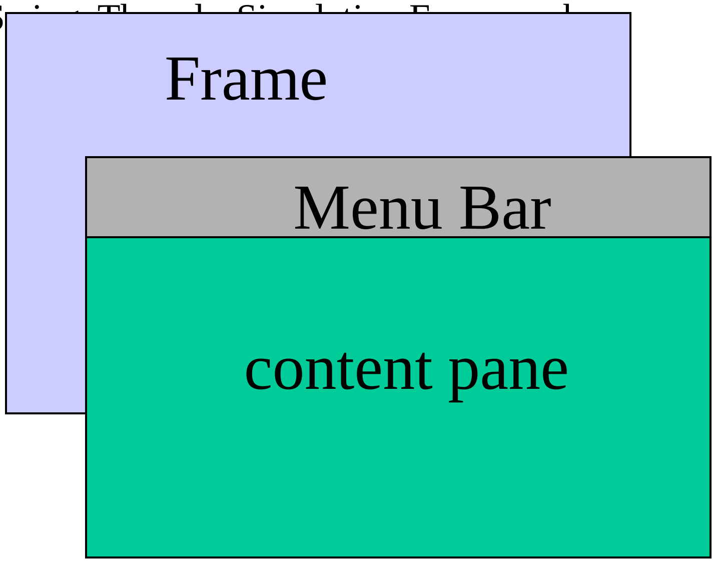
Top level containers are the root of a containment hierarchy.
Interface between the native windowing environment and window manager and java application or applet

JApplet, JFrame and JDialog contain a JRootPane which contains a contentPane container.

components and layout managers must be added and set to the content pane not JApplet, JFrame or JDialog. (else exception thrown)







```
    JFrame frame = new JFrame("frame title");
```

Constructor creates menubar and sets layout managers.

Need to get the content pane to add components to the frame

```
    frame.getContentPane().add(greenLabel, BorderLayout.CENTER);
```

Adding a menu bar to the frame

```
    frame.setJMenuBar(greyMenuBar);
```

Respond to window closing

```
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
Resolve geometry layowt and make JFrame visible
    frame.validate();       // resolves all containment management
    frame.setVisible();     // displays the JFrame
```

Menus

Menus in menubars or as popup

Simple menu setup steps: (order can vary)

1. extend a JFrame for top container
2. instantiate a JMenuBar
3. instantiate a JMenu for every menu choice in the menu bar
4. instantiate each JMenuItem, JRadioButtonMenuItem, or JCheckboxMenuItem choice in the JMenu
    - set JMenuItem icon
    - set mnemonics and/or key accelerators
    - add any submenus (pull right menus)
    - add any item separators

3. add the JMenuItems to JMenu

4a. add a common menuItemListener for all (set of) menu items

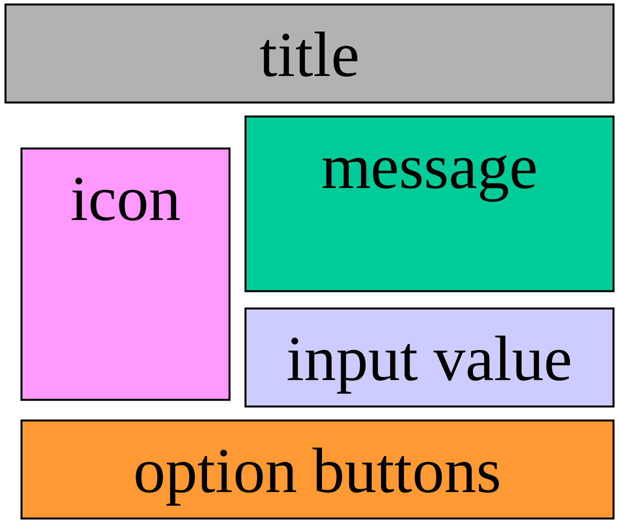
4b. add an ActionListener each menu item.

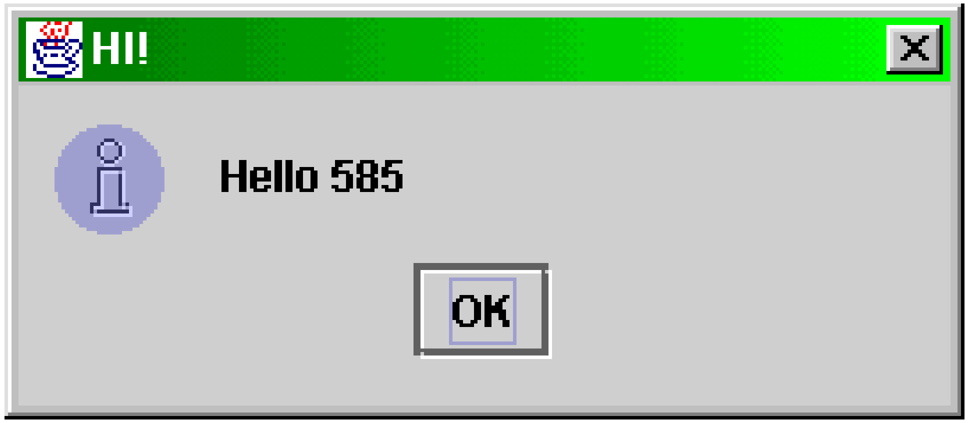
5. add the JMenu to the JMenuBar
6. set the MenuBar in the frame

Dialogs

Swing has many pre-build dialogs – termporary windows

JOptionPane class has prebuilt, message, warning, information, question dialogs with default icons and "ok" button. Also customizable.





```
JOptionPane.showMessageDialog( null, "Hello 585",
   "HI!", JOptionPane.INFORMATION_MESSAGE);
```

<< walkthrough Pipe.java >>

Swing UI elements

Swing Containers

JApplet
JDialog
JFrame
} contentPane

JWindow

JPanel

JScrollPane
JSplitPane
JTtabbedPane
JToolBar

JInternalFrame
JLayeredPane
JRootPane

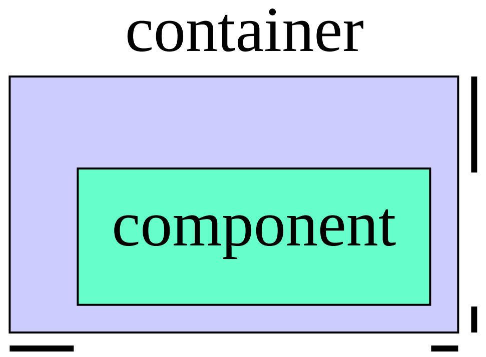
Swing Components

JButton, w/ icon labels
JMenu
JList
JSlider
JTextField, JPasswordField
JComboBox

Displays

JLabel
JProgressBar
JToolTip
JColorChooser
JFileChooser
JTable -- database records
JTextArea, JEditorPane,
JTree

Layout Management



How to place components inside containers.

Containers have insets and default layout managers.

Insets (top, right, left, bottom)
    layout mgrs place components in contain w/o overlaying insets

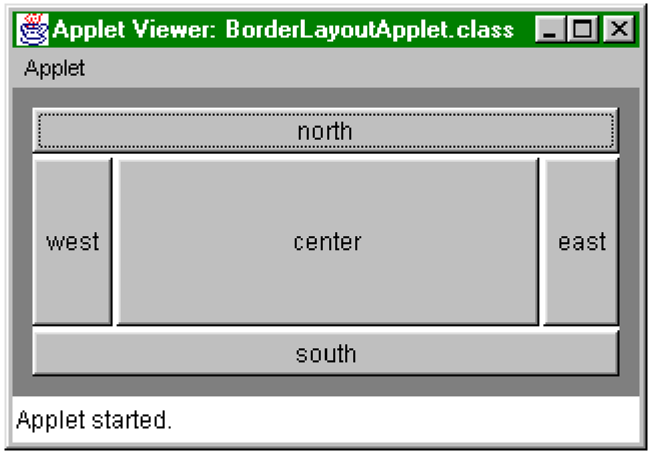
AWT layout managers:

FlowLayout	left to right, top to bottom
BorderLayout	**north, south, east, west, center**
CardLayout	1 panel @ time selected from deck
GridLayout	place on and stretch to row column grid
GridBagLayout	constraint based layout on grid – **avoid** ...

Swing layout manager

BoxLayout	single row or column, w/ glue, rigid fillers
internal managers	used by swing (ScrollPaneLayout JScrollPane)
custom managers	write your own

BorderLayout



default for contentPane
(JApplet, JFrame, JDialog)
uses position constraints

Add components to positions:
north, south, west, east, center
default position is center

```
aContentPane.add(aPanel, BorderLayout.NORTH);
aContentPane.add(aList);
```

Center position grows with resizing and fills all space not used by other positions.

Constructs layout w/o gaps between components

```
new BorderLayout();
```

To set gaps between components

```
new BorderLayout(int Hgap, int Vgap);
```

or

```
setHgap(int)  setVgap(int)
```

JComponent

Components are UI elements that are the members of containers and managed by layout managers. They are the ***visible*** elements of the UI.

Common Components

controls: JButton, JTextField, JSlider

displays: JLabel, JToolTip, JProgressBar

All components have many properties for location, size, color, ... some have an visually interesting properties like icon.

Properties manipulated with set* and get* methods

```
aLabel.setText(new String(“Read me!”));
aString = aTextField.getText();
aLabel.setFont(new Font(“Sans-Serif”, Font.PLAIN, 24));
aButton.setBackground(Color.blue);
aLabel.setIcon(new ImageIcon(“anIcon.gif”);
aLabel.setMinimumSize(new Dimension(width, height));
aButton.setCursor(new Cursor(Cursor.HAND_CURSOR));
```

JLabel, JButton

```
JLabel(“Hi There”);  JLabel(“hi”, JLabel.RIGHT);
JLabel(new ImageIcon(“icon.gif”));
JLabel(“An icon”, new ImageIcon(“icon.gif”),
   JLabel.Center);
```

HTML can be used to specify Strings in labels, buttons, tooltips, ...

JButton

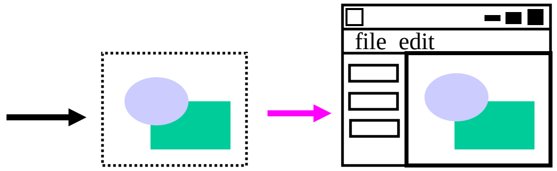
```
... implements ActionListener ...
JButton b1 = new JButton(“Button 1”);   // b2 also ...
b1.setMnemonic(“1”);           // “alt 1”   // b2 also
b1.setEnabled(false);          // initially disabled
b1.addActionListener(this);  // b2 also ...
...
public void actionPerformed(ActionEvent e) {
   if (e.getSource() == b2)   {
      b1.setEnabled(true);
      b2.setEnabled(false); }...
```

Java Graphics

Component and Graphics classes (also Java2D API and Java 3D API)

Drawing graphics has 2 parts / steps
- How to draw -- attributes of the graphic context (Graphics)
  - Color, Font, foreGround, backGround, ...
- What to draw -- a draw request
  - drawLine(...), drawRect(...), fillRect(...), ...

Drawing differs in AWT & Swing



AWT heavyweight components are drawn directly -- not double buffered. This can cause flicker w/ animation & requires double buffering techniques.

Swing lightweight components are double buffered (drawn indirectly) if they are held by JPanel or JRootPane containers.

Graphics

Write a JComponent’s **`paint (Graphics g)`** method for custom drawing.

```
public void paint (Graphics g) {
   g.setColor(new Color (255, 0, 0);
   g.fillRect(25, 25, 100, 20);
   g.drawString(“rgb is:  “ + g.getColor(), 130, 40);
   g.setColor(new Color (0.0f, 1.0f, 0.0f);
   g.fillRect(25, 50, 100, 20);
   g.drawString(“rgb is:  “ + g.getColor(), 130, 40);
   }
```

Usually Graphics g is not known, to call paint(g) directly...

To invoke paint call repaint();

Repaint invokes update(). // Update() called on expose & configure

Update clears the component’s background of any previous draws and calls paint(...).

Java 2D API more shapes, line styles, bufferedImages

Draw requests

All public void. All arguments int unless specified.

```
drawLine( x1, y1, x2, y2);
drawRect( x, y, width, height);
fillRect(...);
clearRect(...); // draws in background
drawRoundRect( x, y, width, height, arcWidth, arcHeight);
fillRoundRect(...);
draw3DRect(x, y, width, height, boolean raised);
drawOval(x, y, width, height);
fillOval(x, y, width, height);
drawArc(x, y, width, height, startAngle, arcAngle);
   angles are degrees.
fillArc(...);
drawPolygon(xPoints[], yPoints[], points); // closed poly
drawPolyLine(xPoints[], yPoints[], points);
drawFillPolygon(xPoints[], yPoints[], points); // closed

public Polygon(xValues[], yValues[], points);
drawPolygon(Polygon p);
fillPolygon(Polygon p);
```

Color

```
Color(int r, int g, int b)            // 0..255
Color(float r, float g, float b);     // 0.0f .. 1.0f

static fields of Color:
Color.orange, pink, cyan, magenta, yellow, black,
white, gray, light gray, dark gray, red, green, blue

Color.orange   Color(255, 200, 0)
Color.pink     Color(255, 175, 175)

public void paintComponent(Graphics g) {
   g.setBackground(Color.black);
   g.setColor = new Color(120, 60, 230);  // ?color?
   g.drawLine(10, 10, 100, 100);
   ... }
```

Using JColorChooser to get a color

```
...
color = JColorChooser.showDialog(...);
if (color == null) color = Color.black; // cancel state
aComponent.setBackground(color);
aComponent.repaint();
```

Mouse Event

Mouse Event Handling: MouseAdapter, MouseMotionAdapter

MouseEvent methods:

```
   int getClickCount(); // count quick consecutive clicks
   int getX(), int getY(); // get point position values
   Point getPoint();
   boolean isPopupTrigger();
   Component getComponent();
   int getWhen();      // returns time stamp
   boolean isAltDown(); isMetaDown();isShiftDown();
```

static SwingUtilities

```
   static boolean isLeftMouseButton(MouseEvent);
   static boolean isMiddleMouseButton(MouseEvent);
   static boolean isRightMouseButton(MouseEvent);
```

Point class encapsulate an X and Y location in a 2D coordinate space (usually a display window).

MouseAdapter

mousePressed(MouseEvent)	button is pressed over component
mouseClicked (MouseEvent)	button released after press w/o drag
mouseReleased (MouseEvent)	button is released
mouseEntered (MouseEvent)	mouse enters component area
mouseExited (MouseEvent)	mouse leaves component area

MouseMotionAdapter

mouseDragged (MouseEvent)	mouse pressed and dragged
mouseMoved(MouseEvent)	mouse moved (not pressed)

Keyboard Event Handling

KeyListener has methods for keyPressed, keyReleased, keyTyped, isMetaDown(), isAltDown ....

Useful in JTextArea objects

Threads

Swing components can be accessed by one thread at a time (usually the event-dispatching thread) . Swing **IS NOT** thread safe.

General usage:

Once a component is realized all code related to the component should be executed in the event-dispatching thread.

Exceptions:

repaint() and revalidate() queue method requests in the event dispatching thread – can be called from other threads

add and remove listeners are thread safe

SwingUtilities that allow threads to place code in event-dispatch thread

```
invokeLater(...)     // does not wait for method to be executed
invokeAndWait(...)   // can deadlock
```

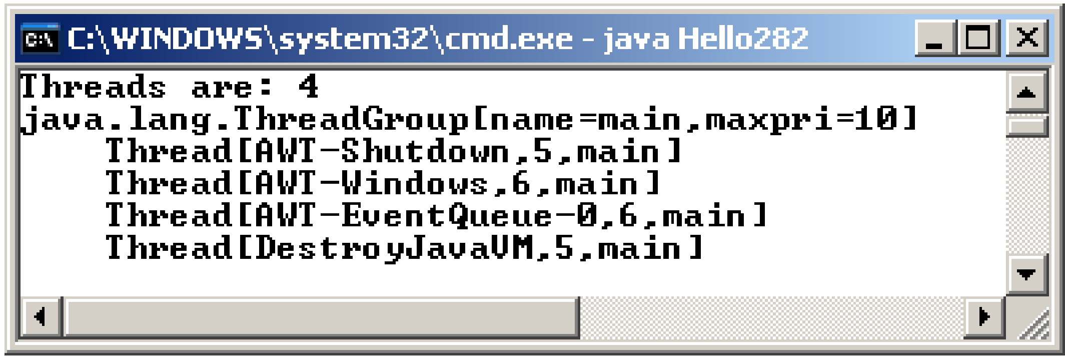
Threads, synchronization

A GUI application has several threads
  [threadName, priority, parentThread]

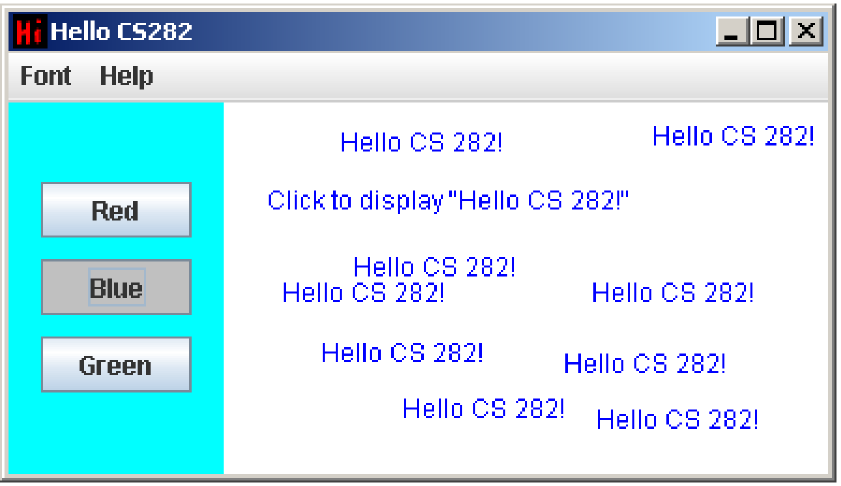
Java GUI events are handled in the AWT-EventQueue

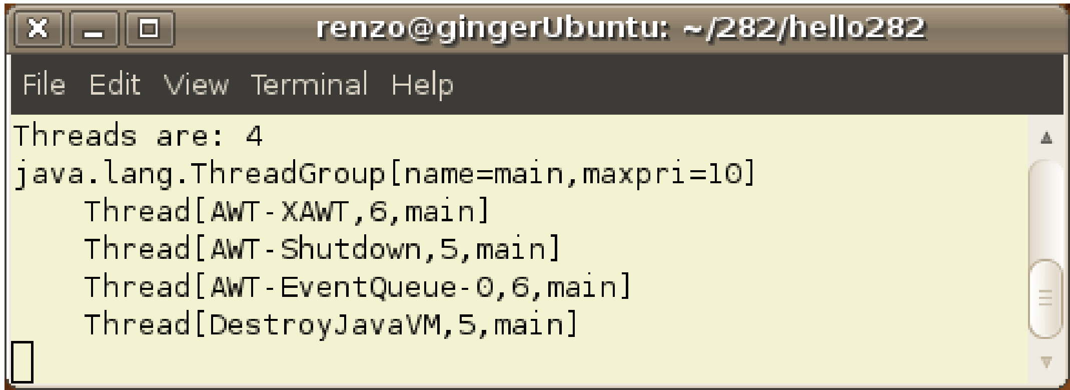
Swing applications can have different threads on different OSs

Synchronization allows threads to control mutually exclusive access to resources (later in notes).



```
Threads are: 4
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-Windows,6,main]
    Thread[AWT-EventQueue-0,6,main]
    Thread[DestroyJavaVM,5,main]
```





```
Threads are: 4
java.lang.ThreadGroup[name=main,maxpri=10]
    Thread[AWT-XAWT,6,main]
    Thread[AWT-Shutdown,5,main]
    Thread[AWT-EventQueue-0,6,main]
    Thread[DestroyJavaVM,5,main]
```

JTable

boolean	checkbox	
Number	label	right
ImageIcon	label	center
Object	label	toString()

Swing has several ADTs

JTable displays model (rows & columns)
    row cells can have renders.

Displays column headers.
    row cell values can be editable.
    can have custom cellEditors -- column editors

Model can be derived from AbstractTableModel if subclass is needed or DefaultTableModel can be instantiated (no subclass needed).

TableModelListener detects changes to data.
    implement **`void tableChanged(TableModelEvent e) {...}`**

DefaultTableModel can manipulate data order

```
moveRow(int start, int end, int to) // from to
```

TableSorter can be use to order the model

JList

Contains with selectable items (see also JComboBox, JTable)

JLists use a ListModel, or array of objects, or vector

JList(ListModel) presents a dynamic set of items

JList(Object[]) or JList(Vector) presents static list items.

static lists == menu of choices (colors, fonts, sizes, zip codes...)

list have a selection model

`SINGLE_SELECTION`	one item
`SINGLE_INTERVAL_SELECTION`	one range adjacent items
`M`**`ULTIPLE_INTERVAL_SELECTION`**	multiple ranges

aList.addListSelectionListener*(this*) requires implementation of method

```
public void valueChanged(ListSelectionEvent e) {....}
```

with single selection **`getSelectedIndex()`** returns the index of the item

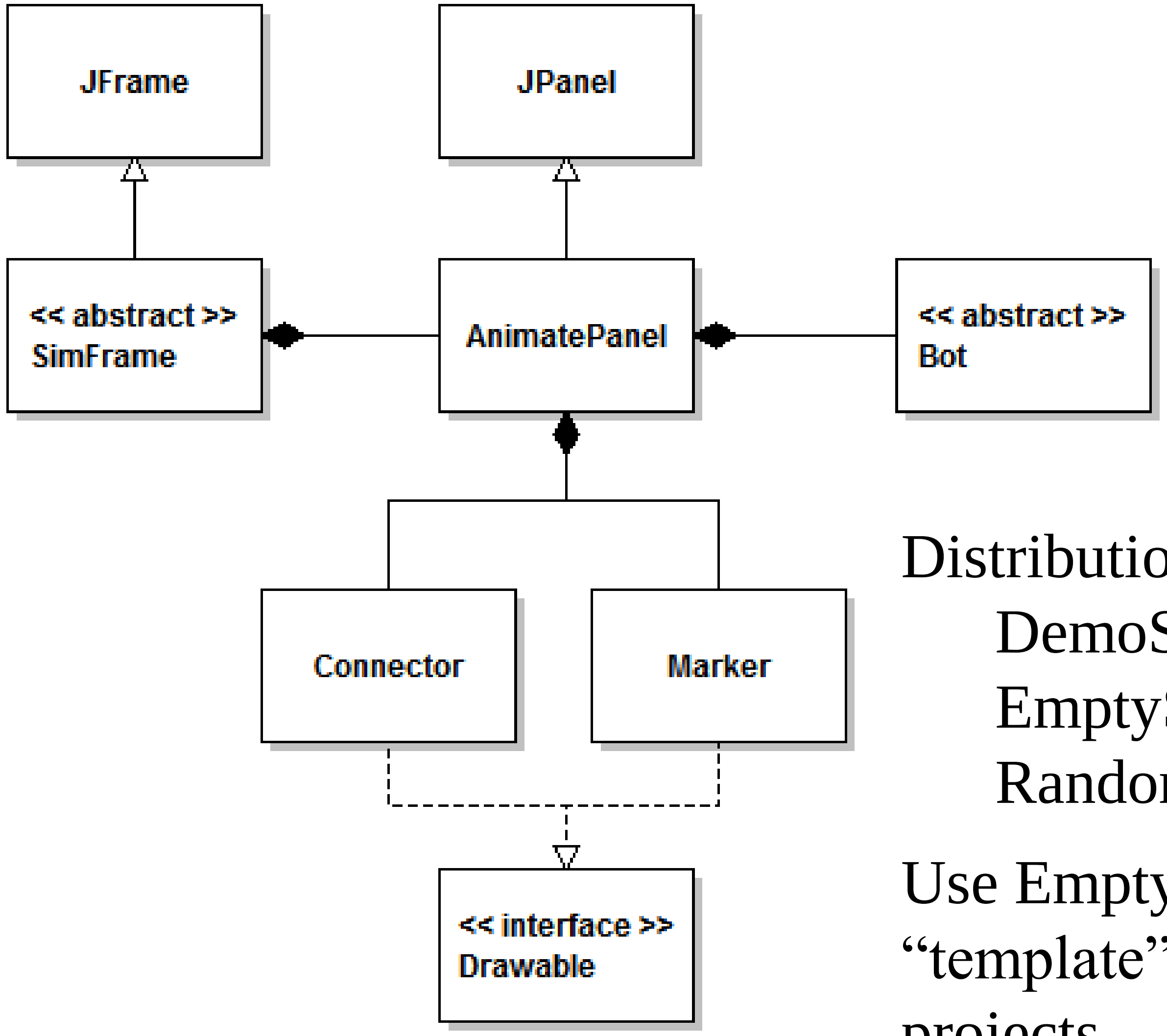
`setSelectedIndex(anInt)` selects item at index anInt

DefaultListModel

```
int getSize();                  // list count
void setSize(int size);

Object get(int index);          // returns index element
Object getElementAt(index);

void setElementAt(Object object, index);
Object set(index, object);         // replace index object
                                   // returns old object

void add(index, object);
void addElement(object);              // add at end of list
void insertElement(object, index);    // add at index
Object remove(index);              // cuts && returns object
void removeElementAt(index);
boolean removeElement(object);     // cuts 1st object found
void removeAllElements();
```

Simulation FrameworkV3

Simulation Framework V3 is a Swing based application with 6 classes for visually simulating algorithms.



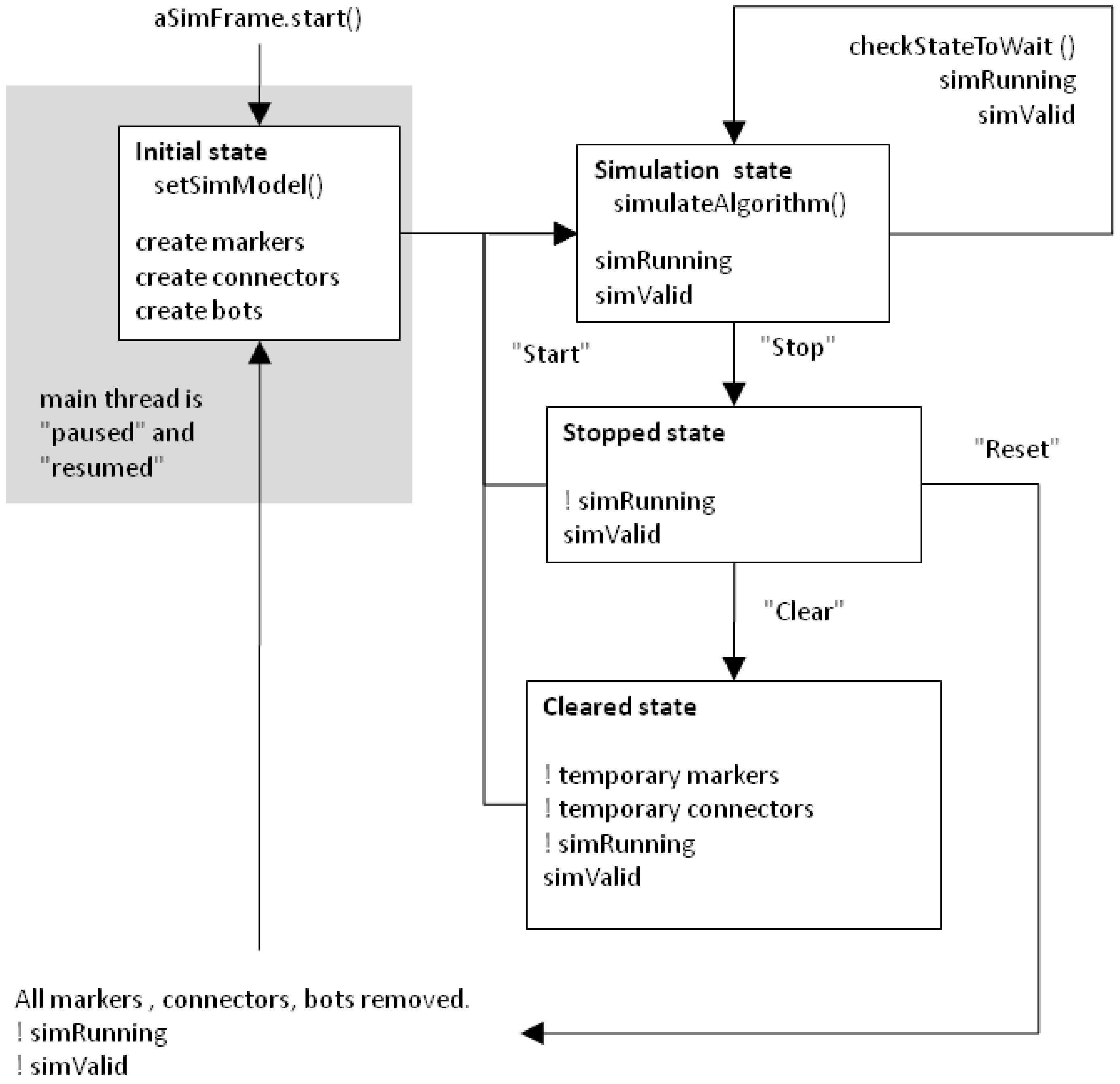
There is javadoc on-line documentation and UML class diagrams for all classes in the /doc subdirectory rooted at index.html

Distribution also has 3 examples:

DemoSimFrame.java
EmptySimFrame.java
RandomBinaryTree.java

Use EmptySimFrame.java as your “template” or “starter file” for your projects.

Simulation states



uses wait() and notifyAll() to “step” algorithm

Synchronized Methods

The class SimFrame.java uses synchronized methods to have a “delay” between algorithm steps and to process mouse events.

To make a method synchronized, simply add the synchronized keyword to its declaration. It has two effects:

First, it is not possible for two invocations of synchronized methods on the same object to interleave. When one thread is executing a synchronized method for an object, all other threads that invoke synchronized methods for the same object block (suspend execution) until the first thread is done with the object.

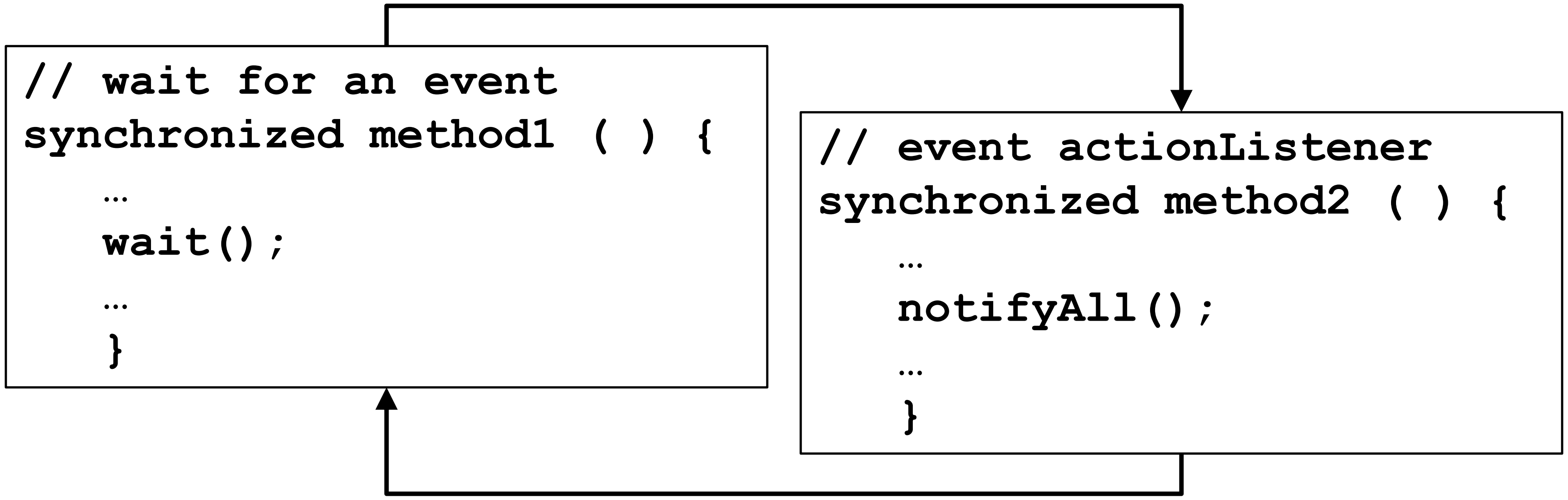
Second, when a synchronized method exits, it automatically establishes a happens-before relationship with *any subsequent invocation* of a synchronized method for the same object. This guarantees that changes to the state of the object are visible to all threads.

wait() && notifyAll()

The Java language includes methods for **thread communication:**

We can call the **`wait()`** method of any Java object, which suspends the current thread. The thread is said to be "waiting on" the given object.

Another thread calls **`notifyAll()`**.
This "wakes up" one (or all) of the threads waiting on that object.



```
// wait for an event
synchronized method1 ( ) {
   …
   wait();
   …
   }
```

```
// event actionListener
synchronized method2 ( ) {
   …
   notifyAll();
   …
   }
```

Using Simulation FrameworkV3

To write a program that uses the Simulation FrameworkV3 classes you do not need to modify any of its 6 classes.

You need to extend 2 of the classes: SimFrame and Bot

ApplicationClass extends SimFrame

- ApplicationClass's constructor creates and defines the
  - JMenuBar menuBar
  - JMenu aboutMenu
    - JMenuItem authorItem
    - JMenuItem usageItem
- ApplicationClass defines
  - setSimModel() to initialize the algorithm
  - simulateAlgorithm() to simulate the algorithm.

Bot must be extended in a "player" or algorithm object class that will define the abstract void move() method.

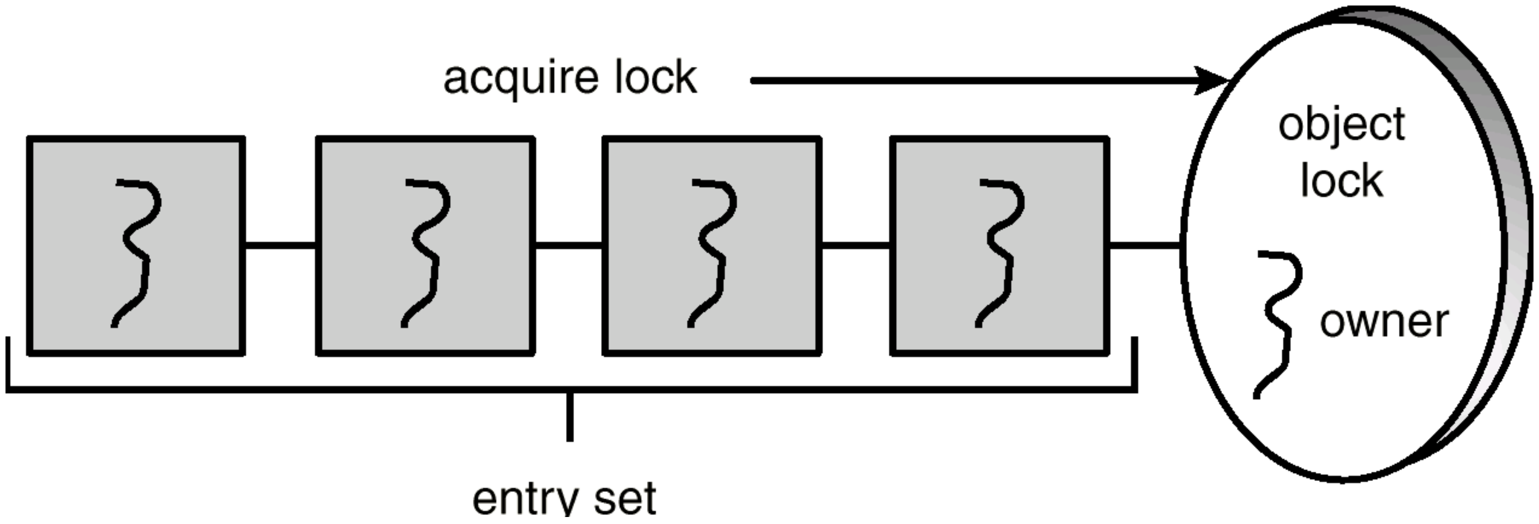
Java Synchronization

Every object has a lock associated with it.

Calling a synchronized method requires "owning" the lock.

If a calling thread does not own the lock (another thread already owns it), the calling thread is placed in the wait set for the object's lock.

The lock is released when a thread exits the synchronized method.



When a thread calls notify(), the following occurs:

- selects an arbitrary thread *T* from the wait set.
- moves *T* to the entry set.
- sets *T* to Runnable.

T can now compete for the object’s lock again.

notify() selects an **arbitrary** thread from the wait set.
This may not be the thread that you want to be selected.

Java does not allow you to specify the thread to be selected.

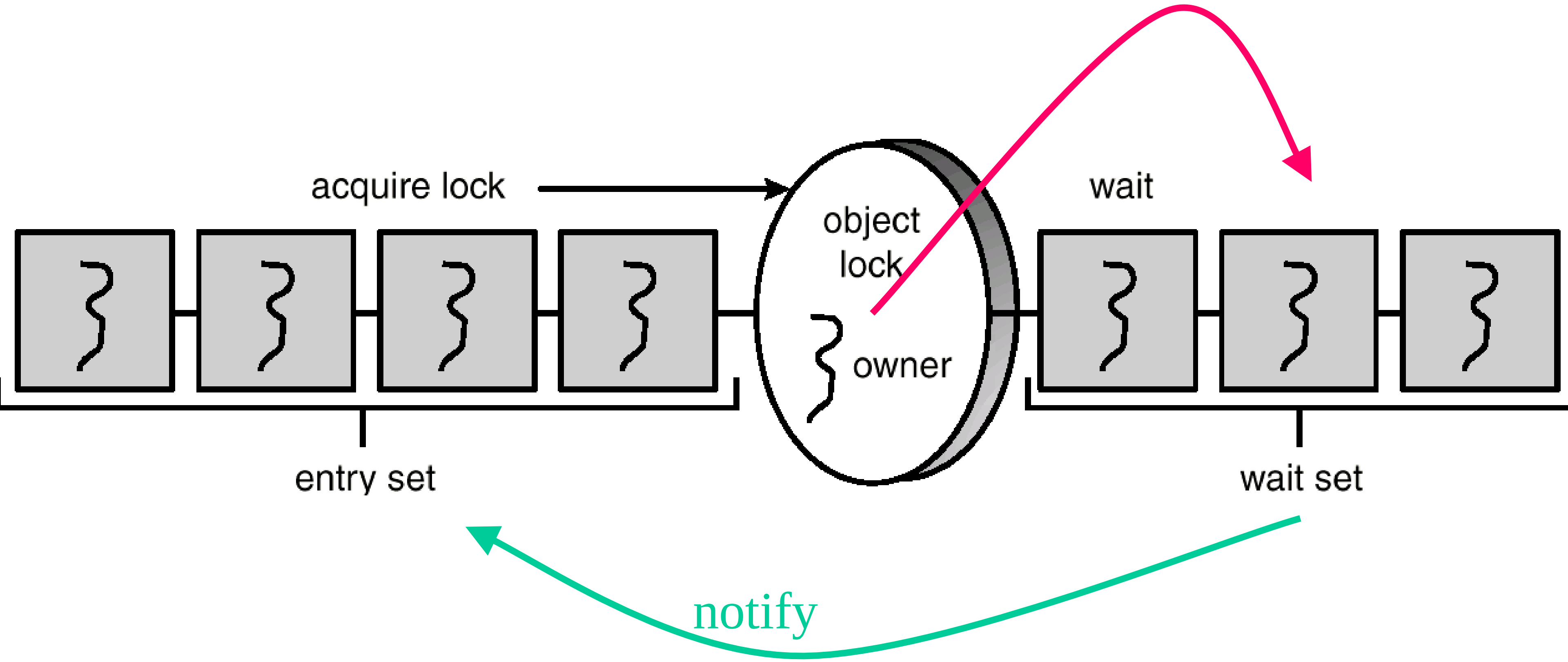
notifyAll() removes ALL threads from the wait set and places them in the entry set. This allows the threads to decide among themselves who should proceed next.

notifyAll() is a conservative strategy that works best when multiple threads may be in the wait set.

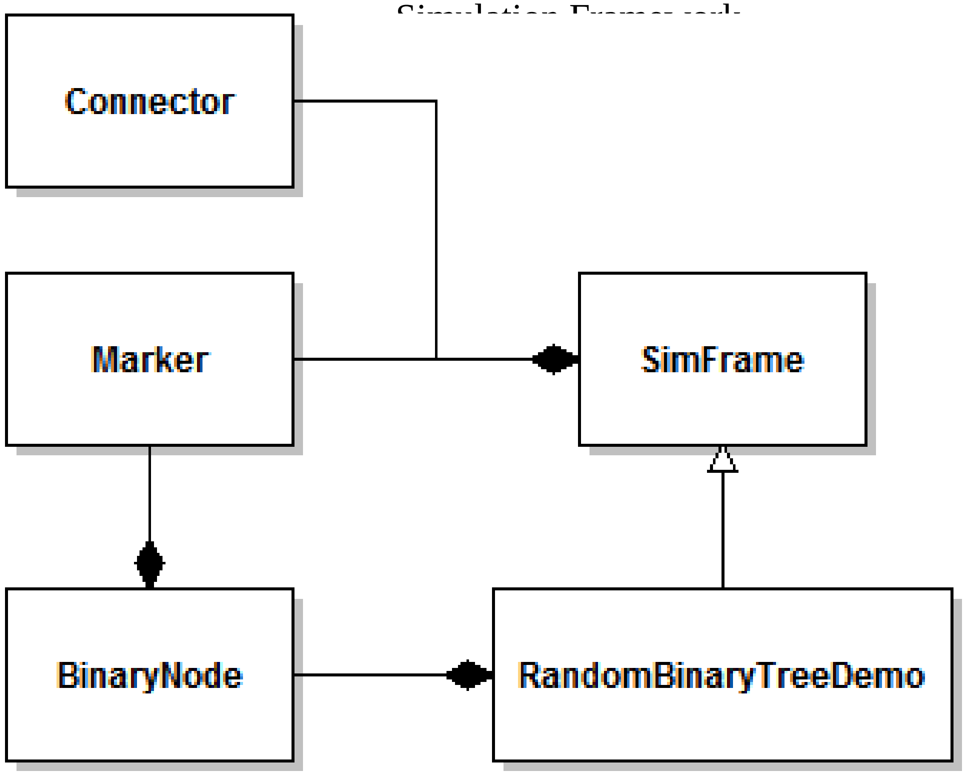
Entry and Wait SetsBalanced binary tree

When a thread calls wait(), the following occurs:

- thread releases the object lock.
- thread state is set to blocked.
- thread is placed in the wait set.



RandomBinaryTreeDemo



Using SimulationFramework

constructor
- create UI controls
- configure “author” and “usuage”
- menuItems
- create int [] key values 0..99

addNodeMarker(...)

inOrder (BinaryNode n, int level, Marker parent)

setSimModel()
- randomize key []
- clear ArrayLists treeVisit, treeLevel

simulateAlgorithm

To use simulateAlgorithm you need to factor the code inside the algorithm's loop out and put it in simulateAlgorithm.

Simulation Framework's aniimation timer becomes the algorithm's loop

```
simulateAlgortrihm
    initialize :  current and newNode BinaryNodes, int level and count
    while ( runnable() )
        if (count <  KEYS ) // building the tree
            checkStateToWait()
            insert next key into binary tree
        else // tree built  validate and get statistics
            setSimRunning(false)
            setModelValid(false)
            animatePanel.setComponentState(...)
            compute statistics using iterators of treeLevel and treeVisit
            checkStateToWait()
```

```
simulateAlgorithm
   initialize the algorithm's variables, state // done once
   boolean state1 = true, state2 = false; ...
   while ( runnable() )
      if (state 1) {
         // statements for 1 pass / step of state 1 actions
         if (state 1 done) {
            state1 = false; state2 = true;  }
         checkStateToWait()  }
      else if (state 2) {
         // statements for 1 pass / step of state 2 actions
         if (state 2 done) {
            state2 = false
            // more states, set next state condition true
            }
         checkStateToWait() }
      ...
      else {
         setSimRunning(false)
         setModelValid(false)
         checkStateToWait()  }
```