Finish Java Collections Framework; GUIs (Graphical User Interfaces)

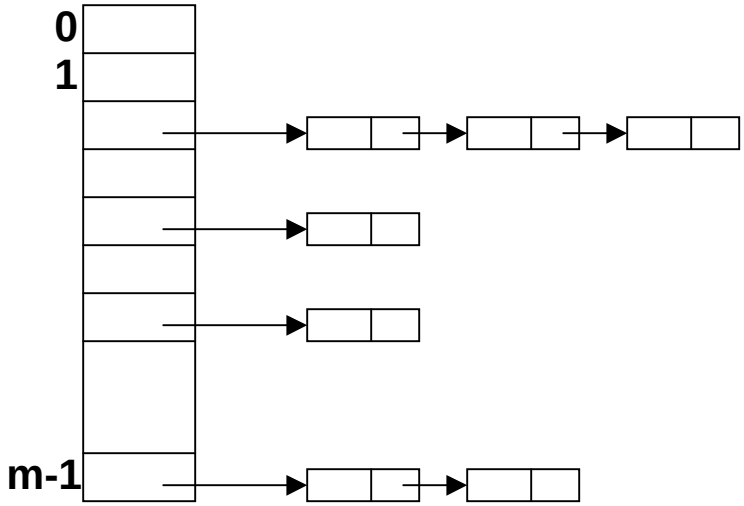
CS211

Fall 2000

Another PQ Implementation

- If there are only a few possible priorities then can use an array of lists
  - Each array position represents a priority (0..m-1 where m is the array size)
  - Each list holds all items that have that priority (treated as a queue)
- One text [Skiena] calls this a *bounded height priority queue*

- Time for add: O(1)
- Time for removeFirst:
  - O(m) in the worst-case
  - Generally, faster



PQ Application: Simulation

- Example: Given a probabilistic model of bank-customer arrival times and transaction times, how many tellers are needed
  - Assume we have a way to generate random inter-arrival times
  - Assume we have a way to generate transaction times
  - Can simulate the bank to get some idea of how long customers must wait

Time-Driven Simulation

- Check at each *tick* to see if any event occurs

Event-Driven Simulation

- Advance clock to next event, skipping intervening *ticks*
- This uses a PQ!

The java.util.Arrays Utility Class

- Provides useful static methods for dealing with arrays
  - sort()
    - Mostly uses QuickSort
    - Uses MergeSort for Object[ ] (it's *stable*)
  - binarySearch()
  - equals()
  - fill()
- These methods are overloaded to work with
  - arrays of each primitive type
  - arrays of Objects
- Methods sort and binarySearch can use the natural order or there is a version of each that can use a Comparator
- There is also a method for viewing an array as a List:

  static List asList (Object[ ] a);
  - Note that the resulting List is *backed by* the array (i.e., changes in the array are reflected in the List and vice versa)

Unmodifiable Collections

- Dangerous version:

public final String suits[ ] = { “Clubs”, “Diamonds”, “Hearts”, “Spades” };

- The final modifier means that suits always refers to the same array, but the array’s elements can be changed
  - suits[0] = “Leisure”;

- Safe version:

private final String theSuits[ ] = { “Clubs”, “Diamonds”, “Hearts”, “Spades” };

public final List suits = Collections.unmodifiableList(Arrays.asList(theSuits));

- The Collections class provides *unmodifiable wrappers*; any methods that would modify the collection throw an UnsupportedOperationException
  - unmodifiableCollection, unmodifiableSet, unmodifiableSortedSet, unmodifiableList
  - unmodifiableMap, unmodifiableSortedMap

The java.util.Collections Utilities

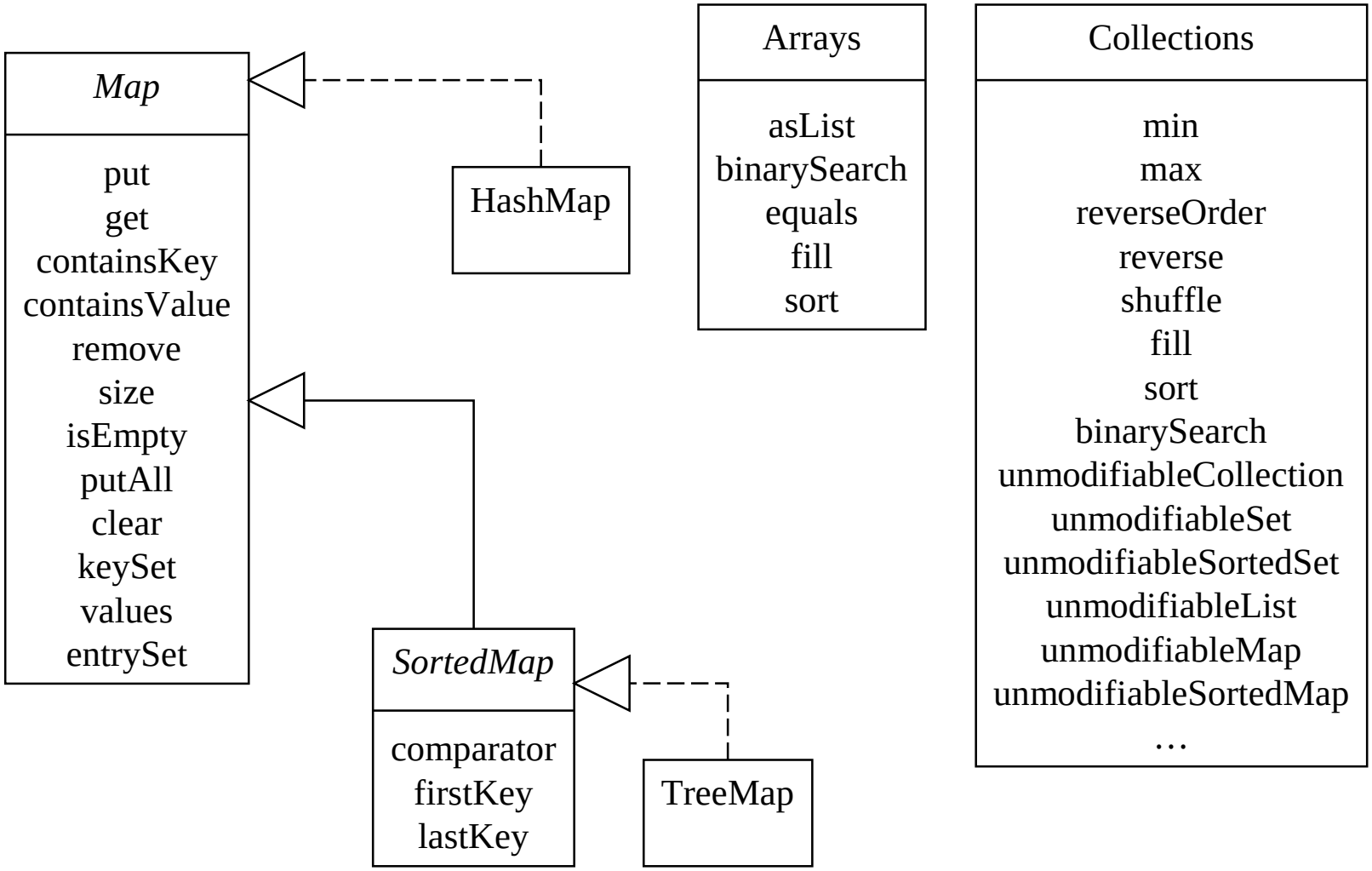
```
public static Object min (Collection c);
public static Object min (Collection c, Comparator comp);
public static Object max (Collection c);
public static Object max (Collection c, Comparator comp);

public static Comparator reverseOrder ( );      // Reverse of natural order

public static void reverse (List list);         // Reverse the list
public static void shuffle (List list);         // Randomly shuffle the list
public static void fill (List list, Object x);  // List is filled with x's

public static void sort (List list);            // Sort using natural order
public static void sort (List list, Comparator comp);
public static void binarySearch (List list, Object key);
public static void binarySearch (List list, Object key, Comparator comp);
...
```

Summary


Map
put
get
containsKey
containsValue
remove
size
isEmpty
putAll
clear
keySet
values
entrySet
HashMap
SortedMap
comparator
firstKey
lastKey
TreeMap
Arrays
asList
binarySearch
equals
fill
sort
Collections
min
max
reverseOrder
reverse
shuffle
fill
sort
binarySearch
unmodifiableCollection
unmodifiableSet
unmodifiableSortedSet
unmodifiableList
unmodifiableMap
unmodifiableSortedMap
…

Graphical User Interfaces

- Layout
  - How items are arranged
  - There are *lots* of predefined GUI items

    JButton, JLabel, JCheckbox, JList, JScrollbar,…
  - You have to write the code that determines layout
  - In Java, you use LayoutManagers to help with layout
- Event Handling
  - An *event* is (generally) a user input or action
  - The JVM (Java Virtual Machine) takes care of generating events

    Button pushed, text typed, mouse clicked,…
  - You have to write the code that determines how your program responds to an event

Swing Components

JButton: a pushbutton that can be clicked by mouse

JCheckbox: can be *on* (true) or *off* (false)

JComboBox: a popup menu of user choices

JLabel: a text label

JList: scrolling list of user-choose-able items

JScrollbar: a scroll bar

JTextField: allows editing of a single line of text

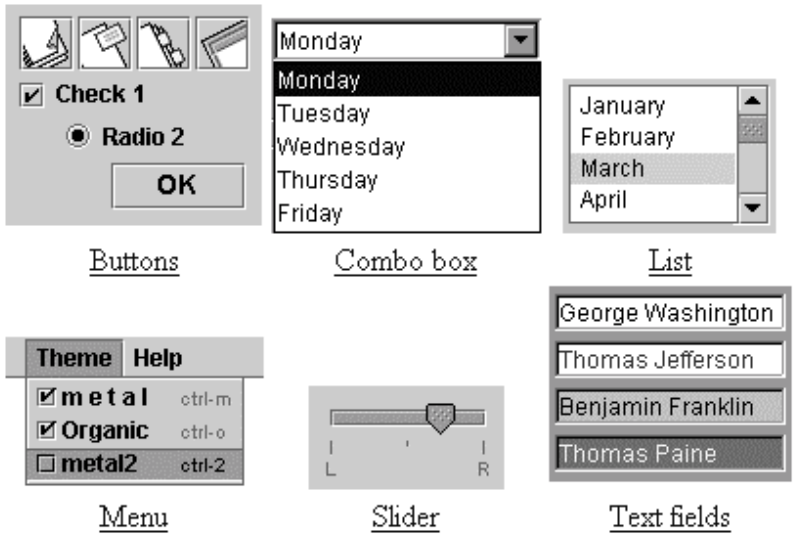
JTextArea: multiline region for displaying and editing text

JPanel: used for containing and grouping components

JDialog: window used for user input

JFrame: top-level window with frame and border

…



javax.swing

- We are using the Swing components instead of the AWT (Abstract Windows Toolkit) components
- The Swing versions are more powerful and support *pluggable look and feel* (your application can look like Windows, Mac, or Motif regardless of underlying platform)
- The AWT components are still supported, but Swing use is recommended (by Sun)
- *Swing* was an internal codename that stuck
- The javax prefix was supposed to correspond to optional extensions, but javax.swing is an official part of Java 1.2 (= Java 2)

Some Components are *Containers*

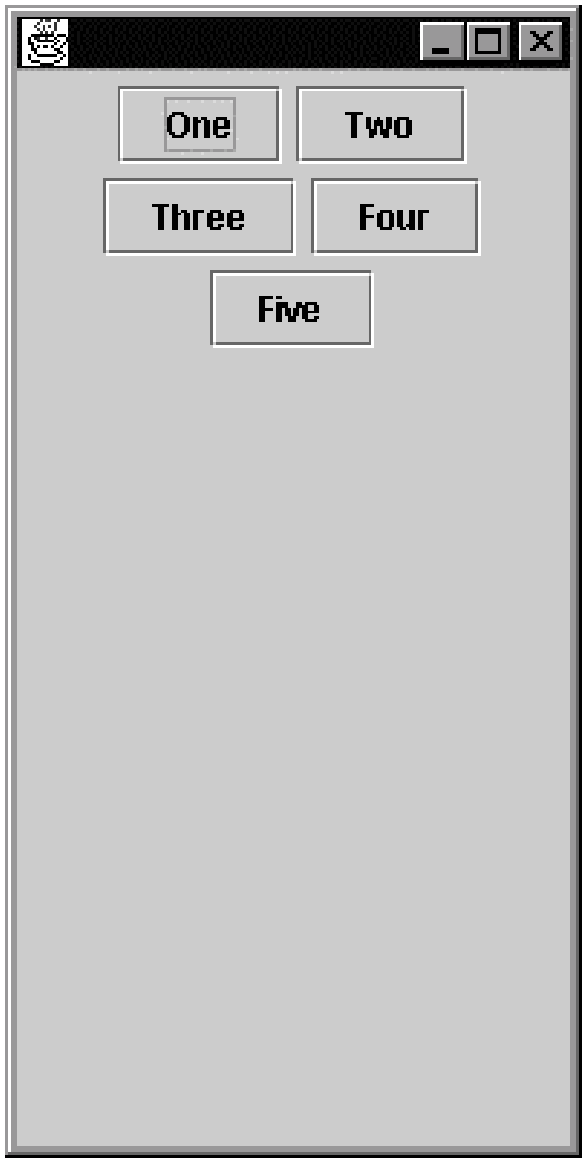
- A *container* is a component that can contain other components
- Since a container is also a component, containers can contain other containers, forming a *containment hierarchy* (*not* the same as the inheritance hierarchy)
- The add(component) method is used to add components to a container
  - Exactly where the component is placed depends on the container's LayoutManager
  - The setLayout(…) method is used to set the container's LayoutManager

Layout Managers

- FlowLayout
  - Use a left-to-right “flow”
  - If one row gets full, start on next row
  - The FlowLayout constructor can take an alignment (default is centered)
    - FlowLayout.LEFT
    - FlowLayout.CENTER
    - FlowLayout.RIGHT
- GridLayout
  - Uses a rectangular grid
  - You specify number of rows and number of columns
    - new GridLayout(3,2);
  - Tries to fill each grid-box
- BorderLayout
  - Uses 5 regions: North, South, East, West, and Center
  - You specify location in add(*component*,*where*);
  - *Where* can be any one of
    - BorderLayout.NORTH
    - BorderLayout.SOUTH
    - BorderLayout.EAST
    - BorderLayout.WEST
    - BorderLayout.CENTER
- Others
  - CardLayout
  - GridBagLayout
  - BoxLayout
  - …

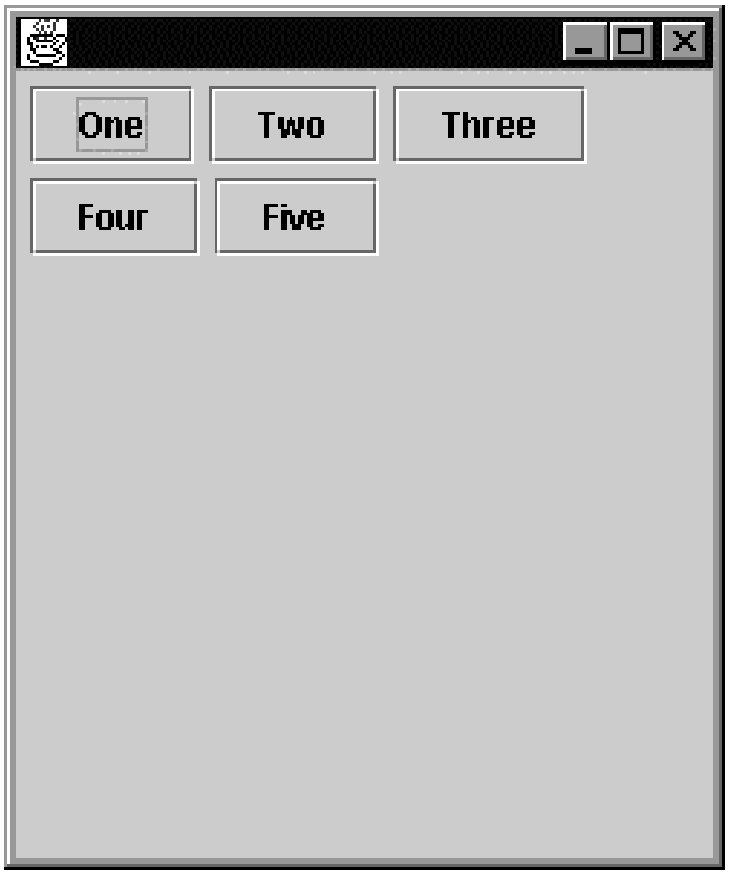
Example: FlowLayout

```
import javax.swing.*;
import java.awt.FlowLayout;
class GUITest {
public static void main (String[ ] args) {
      JFrame frame = new JFrame();
      JPanel panel = new JPanel();
      panel.setLayout(new FlowLayout( ));
      panel.add(new JButton("One"));
      panel.add(new JButton("Two"));
      panel.add(new JButton("Three"));
      panel.add(new JButton("Four"));
      panel.add(new JButton("Five"));
      frame.getContentPane().add(panel);
      frame.setSize(200,400);
      frame.show();
      }
}
```



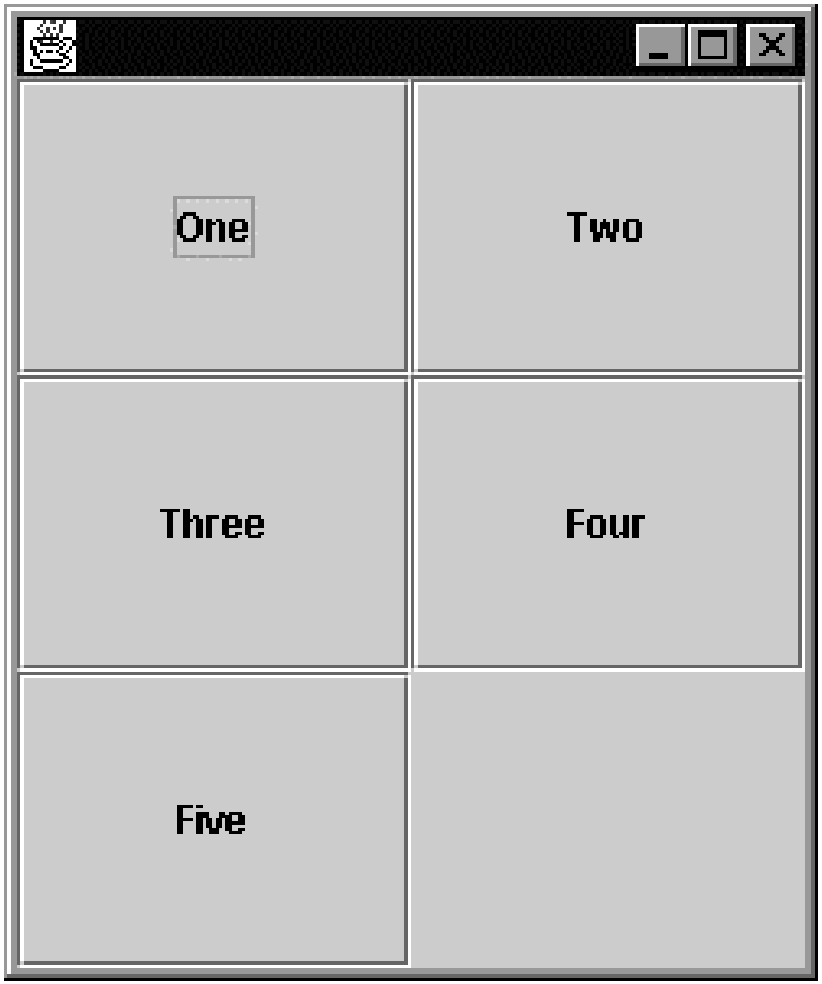
Modified Example: FlowLayout

```
import javax.swing.*;
import java.awt.FlowLayout;
class GUITest {
public static void main (String[ ] args) {
      JFrame frame = new JFrame();
      JPanel panel = new JPanel();
      panel.setLayout(new
            FlowLayout(FlowLayout.LEFT));
      panel.add(new JButton("One"));
      panel.add(new JButton("Two"));
      panel.add(new JButton("Three"));
      panel.add(new JButton("Four"));
      panel.add(new JButton("Five"));
      frame.getContentPane().add(panel);
      frame.setSize(250,300);
      frame.show();
      }
}
```



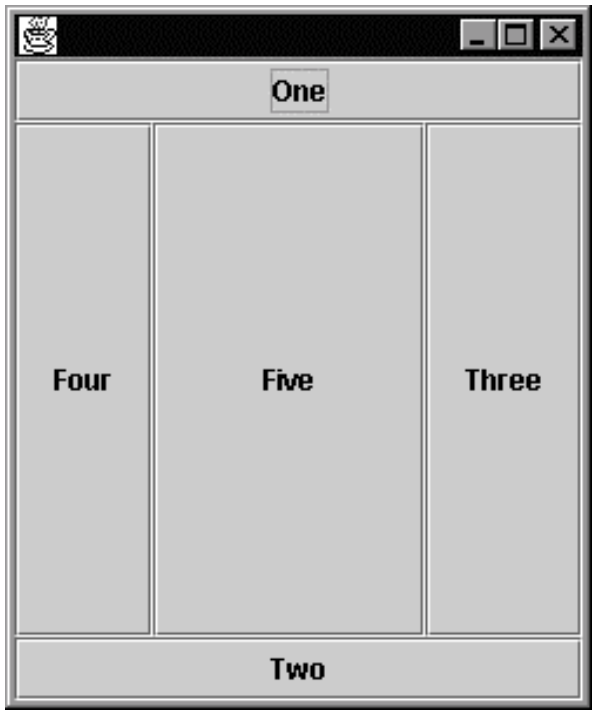
Example: GridLayout

```
import javax.swing.*;
import java.awt.GridLayout;
class GUITest {
public static void main (String[ ] args) {
      JFrame frame = new JFrame();
      JPanel panel = new JPanel();
      panel.setLayout(new GridLayout(3,2));
      panel.add(new JButton("One"));
      panel.add(new JButton("Two"));
      panel.add(new JButton("Three"));
      panel.add(new JButton("Four"));
      panel.add(new JButton("Five"));
      frame.getContentPane().add(panel);
      frame.setSize(250,300);
      frame.show();
      }
}
```



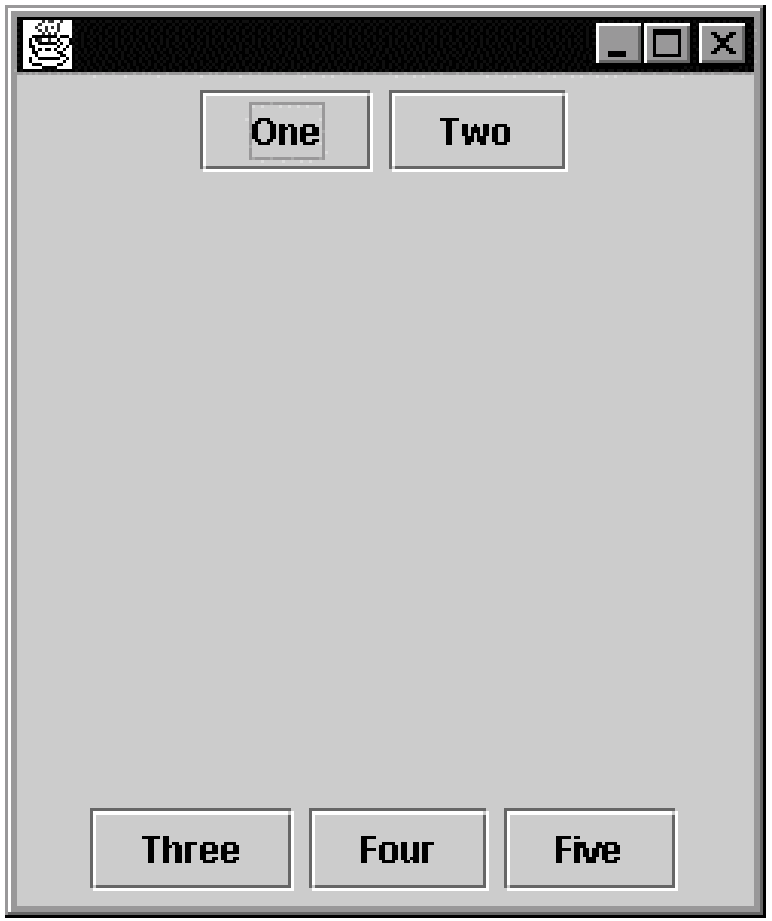
Example: BorderLayout

```
import javax.swing.*;
import java.awt.BorderLayout;
class GUITest {
public static void main (String[ ] args) {
      JFrame frame = new JFrame();
      JPanel panel = new JPanel();
      panel.setLayout(new BorderLayout());
      panel.add(new JButton("One"),BorderLayout.NORTH);
      panel.add(new JButton("Two"),BorderLayout.SOUTH);
      panel.add(new JButton("Three"),BorderLayout.EAST);
      panel.add(new JButton("Four"),BorderLayout.WEST);
      panel.add(new JButton("Five"),BorderLayout.CENTER);
      frame.getContentPane().add(panel);
      frame.setSize(250,300);
      frame.show();
      }
}
```



Using Panels to Group Components

```
public static void main (String[ ] args) {
    JFrame frame = new JFrame();
    JPanel panel = new JPanel();
    JPanel topPanel = new JPanel();
    JPanel botPanel = new JPanel();
    topPanel.add(new JButton("One"));
    topPanel.add(new JButton("Two"));
    botPanel.add(new JButton("Three"));
    botPanel.add(new JButton("Four"));
    botPanel.add(new JButton("Five"));
    panel.setLayout(new BorderLayout());
    panel.add(topPanel,BorderLayout.NORTH);
    panel.add(botPanel,BorderLayout.SOUTH);
    frame.getContentPane().add(panel);
    frame.setSize(250,300);
    frame.show();
    }
```



When an Event Occurs…

- The JVM (Java Virtual Machine) determines the event's *source* and *type*
  - The *source* is the component from which the event originated
  - Each source has certain types of events it can generate
- The JVM looks for one or more *event listeners* that have *registered* with the source
  - An *event listener* is an object that implements one of the *Listener interfaces* in **java.awt.event** or in **javax.swing.event**
  - You *register* an event listener by using one of the component's *addListener* methods
- The JVM creates an *event object* using one of the classes in **java.awt.event** or in **javax.swing.event**
- For each registered event listener, the JVM invokes the listener's event-handling method and passes the event object as the parameter