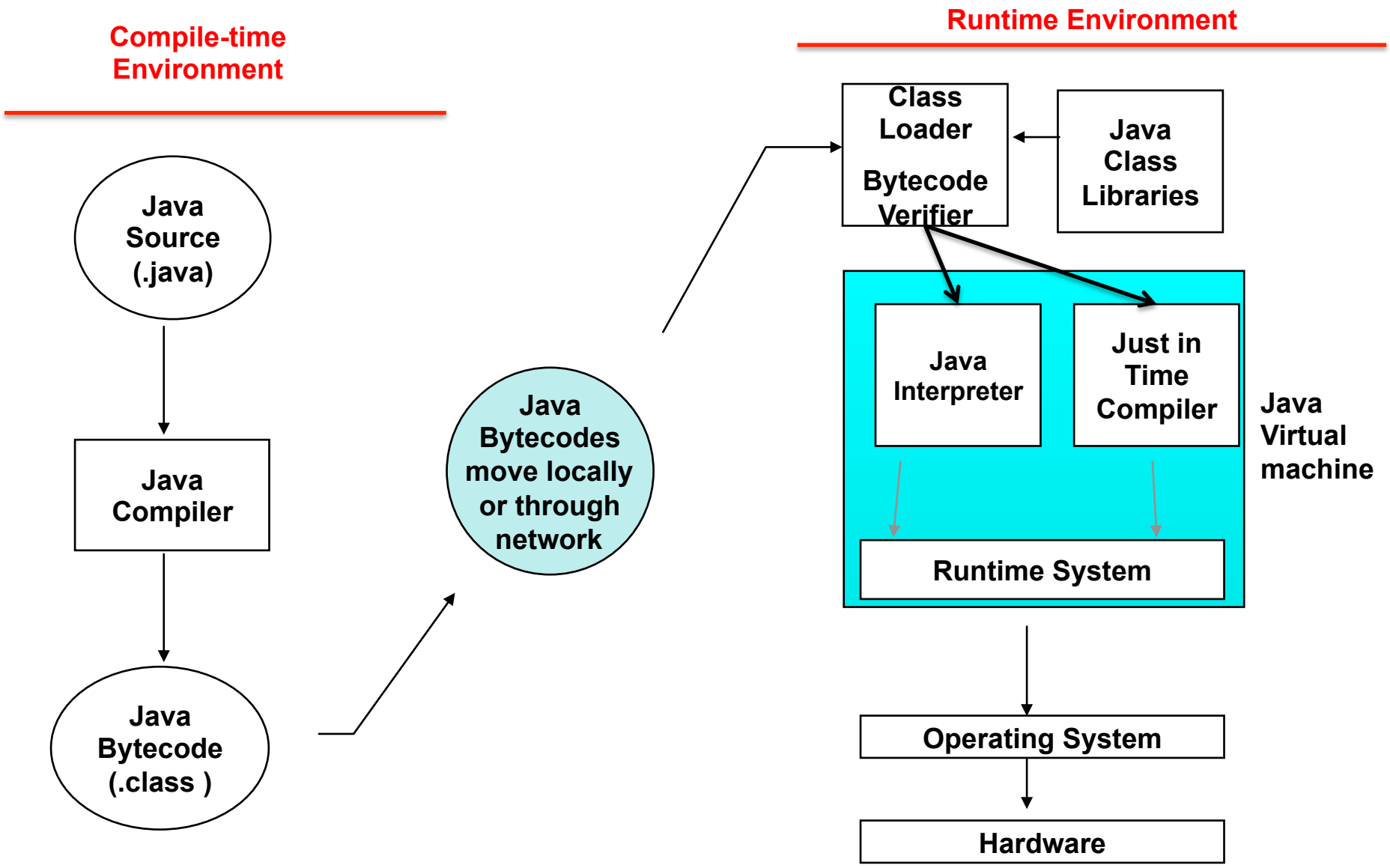
Applets

Java Environment/ Life Cycle of Java Code



Application Vs Applet

Feature	Application	Applet
main() method	Present	Not present
Execution	Requires JRE	Requires a browser like Chrome
Nature	Called as stand-alone application as application can be executed from command prompt	Requires some third party tool help like a browser to execute
Restrictions	Can access any data or software available on the system	cannot access any thing on the system except browser's services
Security	Does not require any security	Requires highest security for the system as they are untrusted

Advantages of Applets

- Execution of applets is easy in a Web browser and does not require any installation or deployment procedure in realtime programming (where as servlets require).
- Writing and displaying (just opening in a browser) graphics and animations is easier than applications.
- In GUI developmenconstructor, size of frame, window closing code etc. are not required (but are required in applications).

Restrictions of Applets

- Applets are required separate compilation before opening in a browser.
- In realtime environment, the bytecode of applet is to be downloaded from the server to the client machine.
- Applets are treated as **untrusted** (as they were developed by unknown people and placed on unknown servers whose trustworthiness is not guaranteed) and for this reason they are not allowed, as a security measure, to access any system resources like file system etc. available on the client system.
- Extra Code is required to communicate between applets using **AppletContext**.

What Applet can't do – Security Limitations

- Applets are treated as **untrusted** because they are developed by somebody and placed on some unknown Web server. When downloaded, they may harm the system resources or steal passwords and valuable information available on the system. As applets are **untrusted**, the browsers come with many security restrictions. Security policies are browser dependent. Browser does not allow the applet to access any of the system resources (applet is permitted to use browser resources, infact, applet execution goes within the browser only).
  - Applets are not permitted to use any system resources like file system as they are untrusted and can inject virus into the system.
  - Applets cannot read from or write to hard disk files.
  - Applet methods cannot be native.
  - Applets should not attempt to create socket connections
  - Applets cannot read system properties
  - Applets cannot use any software available on the system (except browser execution area)
  - Cannot create objects of applications available on the system by composition
  - The JRE throws **SecurityException** if the applet violates the browser restrictions.

Applet Architecture

- Applets permit GUI and handling events. Infact, an applet will be in waiting mode forever expecting some event (input) to occur from the user. Applets are **event driven** and **window-based**. The event is forwarded by the AWT GUI environment (graphics environment) to the applet. The applet takes the event, do some action and return the control back to AWT. Applet does not keep the execution control with it for a long time. If the programmer would like to listen the music continuously or use some banner to display, he must create a separated thread and assign the job to it.
- **Note:** Do not try to take input from keyboard with applets as applets are **window-based** and instead you can create some GUI text field and take input.

Applets

- An applet is a Panel that allows interaction with a Java program
- A applet is typically embedded in a Web page and can be run from a browser
- You need special HTML in the Web page to tell the browser about the applet
- For security reasons, applets run in a sandbox: they have no access to the client's file system

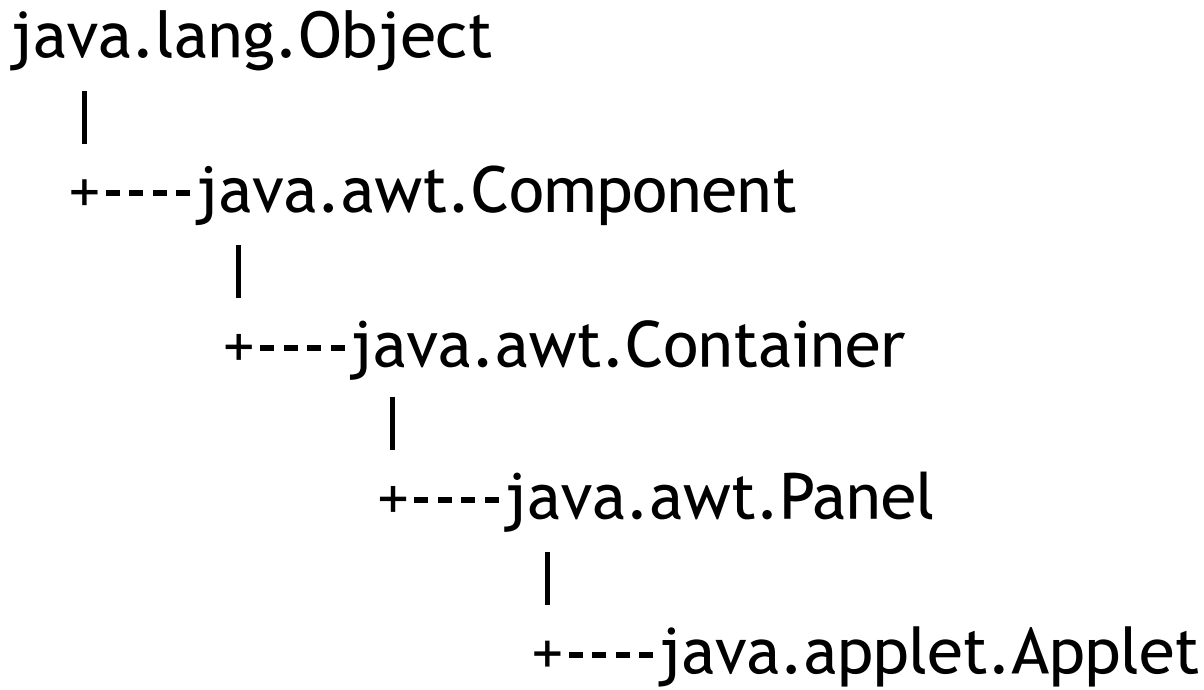
Applet Support

- Most modern browsers support Java 1.4 if they have the appropriate plugin
- In the PC labs, Internet Explorer 5.5 has been updated, but Netscape has not
- The best support isn't a browser, but the standalone program appletviewer
- In general you should try to write applets that can be run with any browser

What an applet is

- You write an applet by extending the class `Applet`
- `Applet` is just a class like any other; you can even use it in applications if you want
- When you write an applet, you are only writing *part* of a program
- The browser supplies the `main` method

The genealogy of Applet



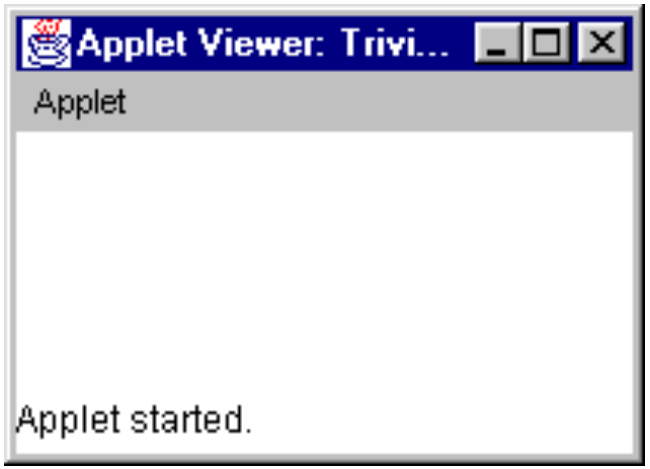
The simplest possible applet

TrivialApplet.java

```
import java.applet.Applet;
public class TrivialApplet extends Applet { }
```

TrivialApplet.html

```
<applet
    code="TrivialApplet.class”
    width=150 height=100>
</applet>
```



Applet methods

public void init ()

public void start ()

public void stop ()

public void destroy ()

public void paint (Graphics)

Also:

public void repaint()

public void update (Graphics)

public void showStatus(String)

public String getParameter(String)

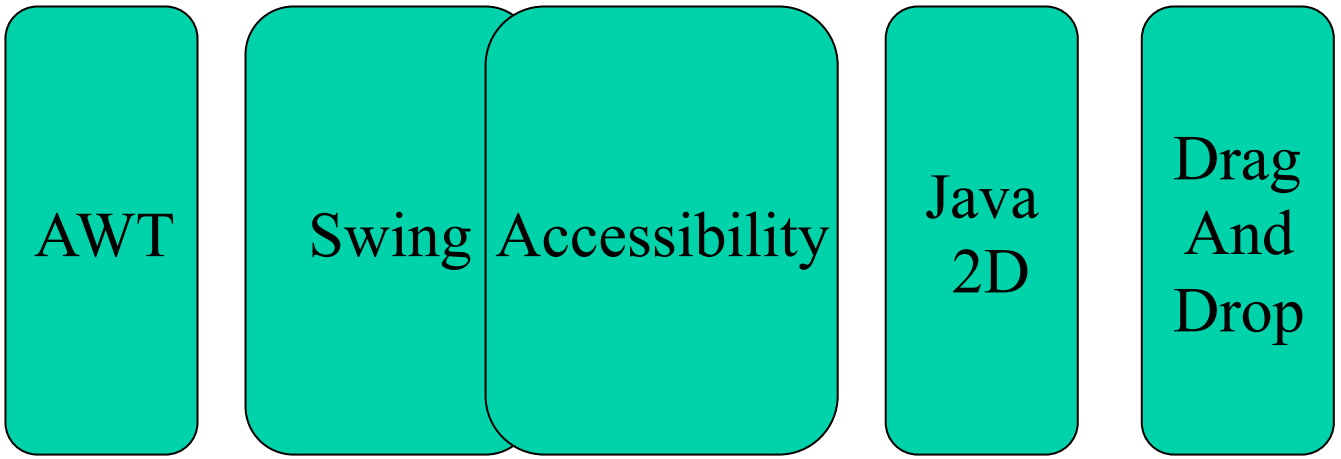
Abstract Window Toolkit

The Abstract Windowing Toolkit

- Since Java was first released, its user interface facilities have been a significant weakness
  - The Abstract Windowing Toolkit (AWT) was part of the JDK form the beginning, but it really was not sufficient to support a complex user interface
- JDK 1.1 fixed a number of problems, and most notably, it introduced a new event model. It did not make any major additions to the basic components

Java Foundation Classes

- In April 1997, JavaSoft announced the Java Foundation Classes (JFC).
  - a major part of the JFC is a new set of user interface components called Swing.



Swing

- The Swing classes are used to build GUIs
  - Swing does not stand for anything
  - Swing is built on top of the 1.1/1.2 AWT libraries
- Swing makes 3 major improvements on the AWT
  - does not rely on the platform's native components
  - it supports "Pluggable Look-and-Feel" or PLAF
  - it is based on the Model-View-Controller (MVC)

GUI Packages

- AWT
  - java.awt
  - java.awt.color
  - java.awt.datatransfer
  - java.awt.event
  - java.awt.font
  - java.awt.geom
  - java.awt.image
  - ...
- Swing
  - javax.accessibility
  - javax.swing
  - javax.swing.colorchooser
  - javax.swing.event
  - javax.swing.filechooser
  - javax.swing.plaf
  - javax.swing.table
  - javax.swing.text.html
  - javax.swing.tree
  - ...

Components

- A GUI consists of different graphic Component objects which are combined into a hierarchy using Container objects.
- Component class
  - An abstract class for GUI components such as menus, buttons, labels, lists, etc.
- Container
  - An abstract class that extends Component. Containers can hold multiple components.

Weighing Components

- Sun makes a distinction between *lightweight* and *heavyweight* components
  - Lightweight components are not dependent on native peers to render themselves. They are coded in Java.
  - Heavyweight components are rendered by the host operating system. They are resources managed by the underlying window manager.

Heavyweight Components

- Heavyweight components were unwieldy for two reasons
  - Equivalent components on different platforms do not necessarily act alike.
  - The look and feel of each component was tied to the host operating system
- Almost all Swing components are lightweight except
  - JApplet, JFrame, JDialog, and JWindow

Additional Swing Features

- Swing also provides
  - A wide variety of components (tables, trees, sliders, progress bars, internal frame, …)
  - Swing components can have *tooltips* placed over them.
  - Arbitrary keyboard events can be bound to components.
  - Additional debugging support.
  - Support for parsing and displaying HTML based information.

Applets versus Applications

- Using Swing it is possible to create two different types of GUI programs
  - Standalone applications
    - Programs that are started from the command line
    - Code resides on the machine on which they are run
  - Applets
    - Programs run inside a web browser
    - Code is downloaded from a web server
    - JVM is contained inside the web browser
    - For security purposes Applets are normally prevented from doing certain things (for example opening files)
- For now we will write standalone applications

Three Types of GUI Classes

1. **Containers**

   JFrame, JPanel, JApplet

2. **Components**

   JButton, JTextField, JComboBox, JList, etc.

3. **Helpers**

   Graphics, Color, Font, Dimension, etc.

JFrames

- A **JFrame** is a Window with all of the adornments added.
  - **JFrame** inherits from **Frame, Window, Container, Component,** and **Object**
- A **JFrame** provides the basic building block for screen-oriented applications.

```
JFrame win = new JFrame( “title” );
```

Creating a `JFrame`

```
import javax.swing.*;

public class SwingFrame {
    public static void main( String args[] ) {
        JFrame win = new JFrame( "My First GUI Program" );

        win.setVisible(true);
    }
} // SwingFrame
```

JFrame

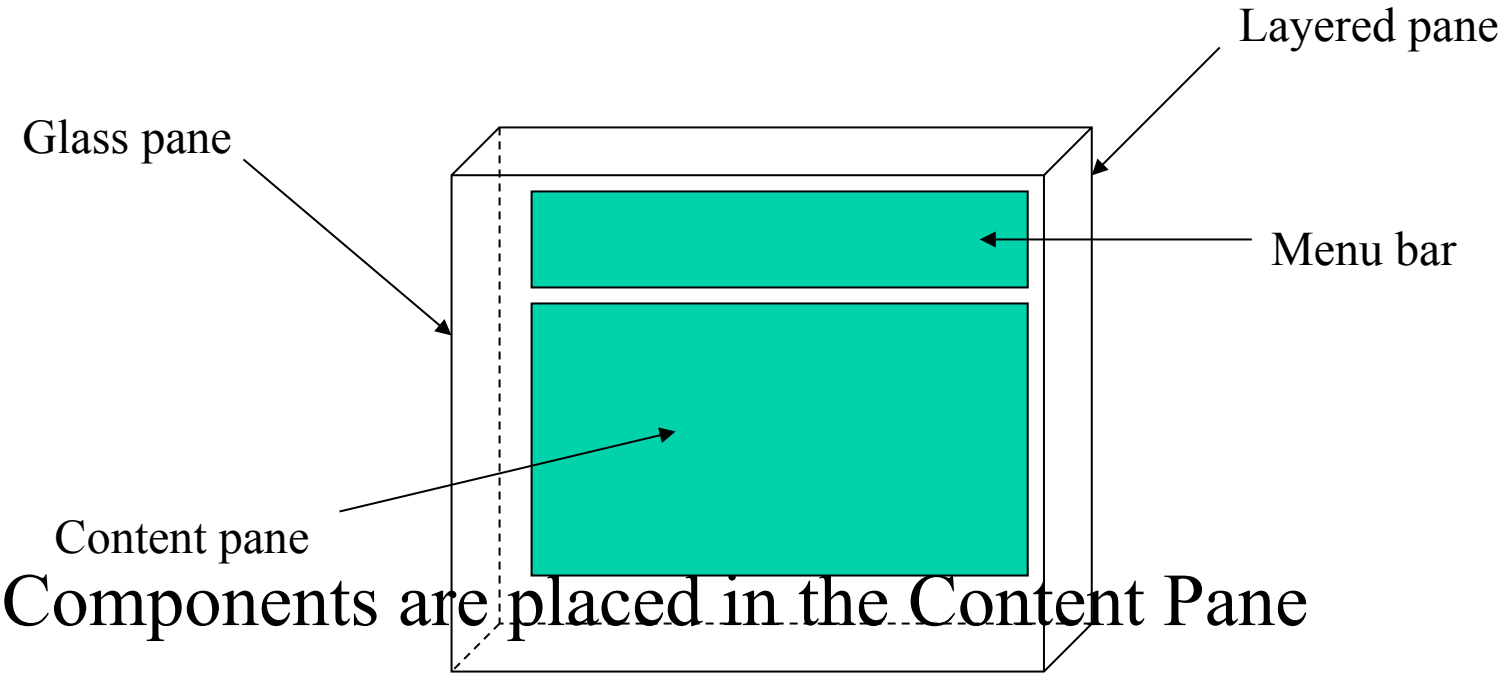
- Sizing a Frame
  - You can specify the size
    - Height and width given in pixels
    - The size of a pixel will vary based on the resolution of the device on which the frame is rendered
    - Usually one uses an instance of the Dimension class.
  - The method **`pack()`** will set the size of the frame automatically, based on the size of the components contained in the content pane
    - Note that pack does not look at the title bar
    - Sometimes pack() does not work as you might expect; try it and see.

Creating a `JFrame`

```
import javax.swing.*;
import java.awt.*;

public class SwingFrame {

    static Dimension windowSize = new Dimension( 250, 150 );

    public static void main( String args[] ) {
        JFrame win = new JFrame( "My First GUI Program" );
        win.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        win.setSize( windowSize );
        win.setVisible(true);
    }
} // SwingFrame
```

JFrame

- **JFrame**s have several panes:



- Components are placed in the Content Pane

Swing Components

- **`JComponent`**
  - **`JComboBox, JLabel, JList, JMenuBar, JPanel, JPopupMenu, JScrollBar, JScrollPane, JTable, JTree, JInternalFrame, JOptionPane, JProgressBar, JRootPane, JSeparator, JSlider, JSplitPane, JTabbedPane, JToolBar, JToolTip, Jviewport, JColorChooser, JTextComponent,` …**

JLabels

- **JLabels** are components that you can fill with text.
- When creating a label you can specify the initial value and the alignment you wish to use within the label.
- You can use **getText()** and **setText()** to get and change the value of the label.

```
lbl = new JLabel( ”text", JLabel.RIGHT );
```

Hello World

```
import javax.swing.*;

public class SwingFrame {
    public static void main( String args[] ) {
        JFrame win = new JFrame( "My First GUI Program" );

        JLabel label = new JLabel( "Hello World" );

        win.getContentPane().add( label );

        win.setVisible(true);
    }
} // SwingFrame
```

JButtons

- **JButton** extends **Component** , displays a string, and delivers an **ActionEvent** for each mouse click.
- Normally buttons are displayed with a border
- In addition to text, **JButtons** can also display icons

```
button = new JButton( ”text“ );
```

Buttons

```
import javax.swing.*;

public class SwingFrame {
    public static void main( String args[] ) {
        JFrame win = new JFrame( "My First GUI Program" );

        JButton button = new JButton( "Click Me!!" );

        win.getContentPane().add( button );

        win.setVisible(true);
    }
} // SwingFrame
```

Layout Manager

- Layout Manager
  - An interface that defines methods for positioning and sizing objects within a container. Java defines several default implementations of **`LayoutManager`**.
- Geometrical placement in a Container is controlled by a **`LayoutManager`** object

Components, Containers, and Layout Managers

- Containers may contain components (which means containers can contain containers!!).
- All containers come equipped with a layout manager which positions and shapes (lays out) the container's components.
- Much of the action in Swing occurs between components, containers, and their layout managers.

Layout Managers

- Layouts allow you to format components on the screen in a platform-independent way
- The standard JDK provides many classes that implement the **`LayoutManager`** interface, including:
  - **`FlowLayout`**
  - **`GridLayout`**
  - **`BorderLayout`**
  - **`BoxLayout`**
  - **`CardLayout`**
  - **`OverlayLayout`**
  - **`GridBagLayout`**

Changing the Layout

1. To change the layout used in a container you first need to create the layout.
2. Then you invoke the **`setLayout()`** method on the container to use the new layout.

```
JPanel p = new JPanel() ;
p.setLayout( new FlowLayout() );
```

- The layout manager should be established before any components are added to the container

FlowLayout

- **FlowLayout** is the default layout for the **JPanel** class.
- When you add components to the screen, they flow left to right (centered) based on the order added and the width of the screen.
- Very similar to word wrap and full justification on a word processor.
- If the screen is resized, the components' flow will change based on the new width and height

Flow Layout

```
import javax.swing.*;
import java.awt.*;

public class ShowFlowLayout {
    public static void main( String args[] ) {
        JFrame win = new JFrame( "My First GUI Program" );
        win.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        win.getContentPane().setLayout( new FlowLayout() );

        for ( int i = 0; i < 10; i++ ) {
            win.getContentPane().add(
                new JButton( String.valueOf( i ) ) );
        }

        win.setVisible(true);
    }
} // ShowFlowLayout
```

FlowLayout


My First GUI Program
0
1
2
3
4
5
6
7
8
9


My First GUI Program
0
1
2
3
4
5
6
7
8
9


My...
0
1
2
3
4
5
6
7
8
9

GridLayout

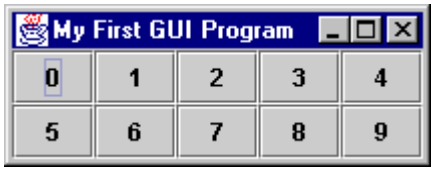
- Arranges components in rows and columns
  - If the number of rows is specified
    - columns = number of components / rows
  - If the number of columns is specified
    - Rows = number of components / columns
  - The number of columns is ignored unless the number of rows is zero.
- The order in which you add components matters
  - Component 1 → (0,0), Component 2 → (0,1), …...
- Components are resized to fit the row-column area

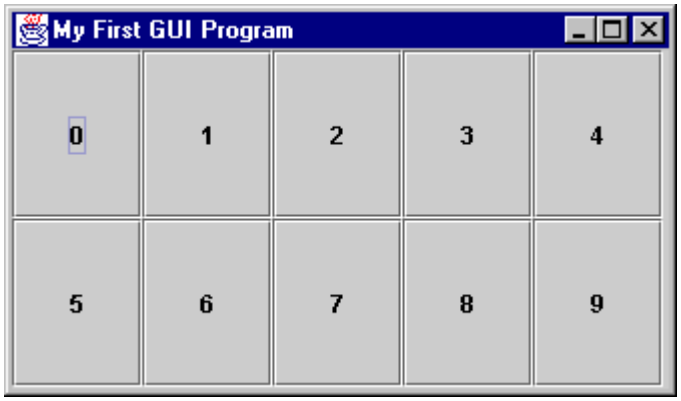
Grid Layout

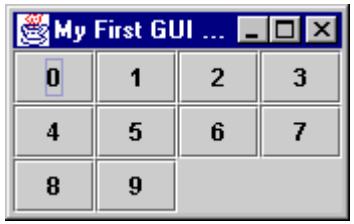
```
import javax.swing.*;
import java.awt.*;

public class ShowGridLayout {
    public static void main( String args[] ) {
        JFrame win = new JFrame( "My First GUI Program" );
        win.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        win.getContentPane().setLayout( new GridLayout( 2, 0 ) );

        for ( int i = 0; i < 10; i++ ){
            win.getContentPane().add(
                new JButton( String.valueOf( i ) ) );
        }

        win.setVisible(true);
    }
} // ShowGridLayout
```

GridLayout

`gridLayout( 2, 4 )`

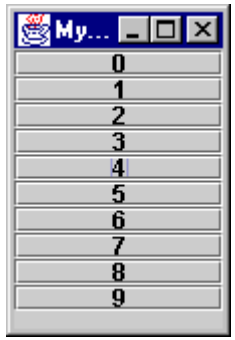






`gridLayout( 0, 4 )`

`gridLayout( 4, 4 )`



`gridLayout( 10, 10 )`

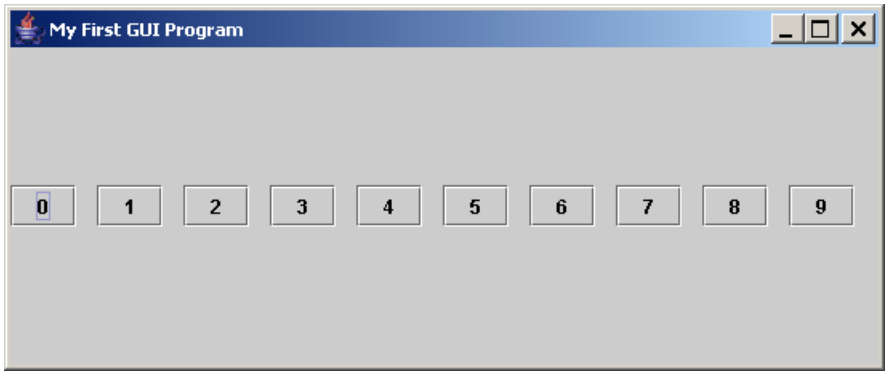
`BoxLayout`

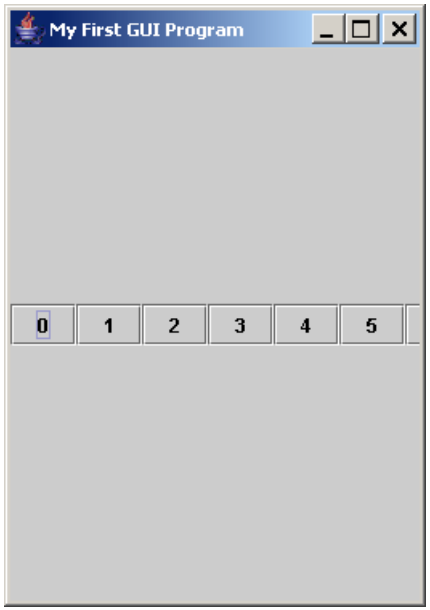
- **`BoxLayout`** provides an easy way to lay out components horizontally or vertically.
- Components are added in order.
- **`BoxLayout`** attempts to arrange components at their
  - *preferred widths* (for horizontal layout) or
  - *preferred heights* (for vertical layout).
- Static methods in **`Box`** class are available for "glue" and "struts."

`BoxLayout` example

```
import javax.swing.*;
import java.awt.*;

public class ShowBoxLayout {
    public static void main( String args[] ) {
        JFrame win = new JFrame( "My First GUI Program" );
        win.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        win.getContentPane().setLayout(
                new BoxLayout( win.getContentPane(), BoxLayout.X_AXIS ) );

        for ( int i = 0; i < 10; i++ ){
            win.getContentPane().add( new JButton( String.valueOf( i ) ) );
            win.getContentPane().add( Box.createHorizontalGlue() );
        }

        win.pack();
        win.setVisible(true);
    }
} // ShowBoxLayout
```

BoxLayout





Note that components retain their preferred size.

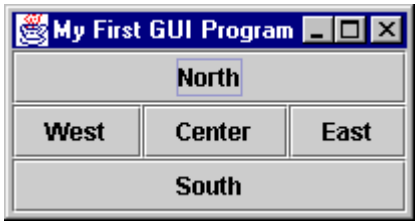
BorderLayout

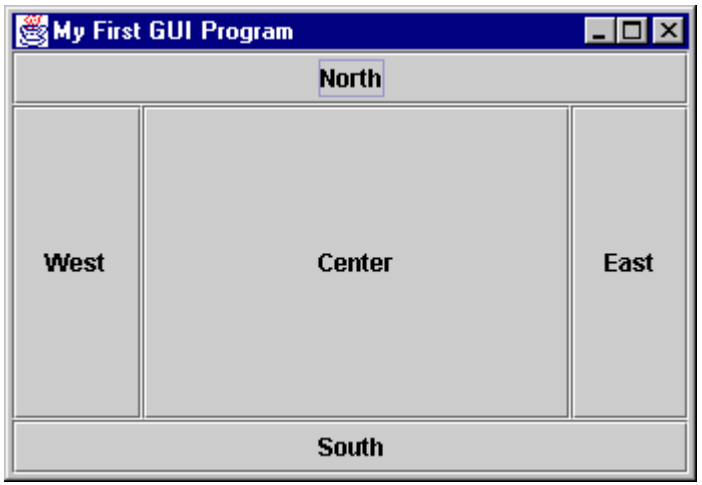
- `BorderLayout` provides 5 areas to hold components. These are named after the four different borders of the screen, North, South, East, West, and Center.
- When a Component is added to the layout, you must specify which area to place it in. The order in which components are added is not important.
- The center area will always be resized to be as large as possible

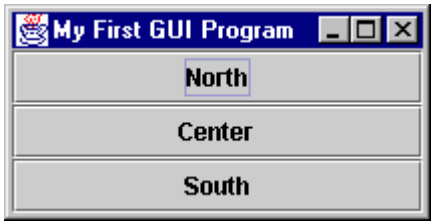
BorderLayout

```
import javax.swing.*;
import java.awt.*;

public class ShowBorderLayout {
    public static void main( String args[] ) {
        JFrame win = new JFrame( "My First GUI Program" );
        win.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        Container content = win.getContentPane();
        content.setLayout( new BorderLayout() );
        content.add( BorderLayout.NORTH,  new JButton( "North" ) );
        content.add( "South",  new JButton( "South" ) );
        content.add( "East",   new JButton( "East" ) );
        content.add( "West",   new JButton( "West" ) );
        content.add( "Center", new JButton( "Center" ) );

        win.setVisible(true);
    }
} // ShowBorderLayout
```

BorderLayout



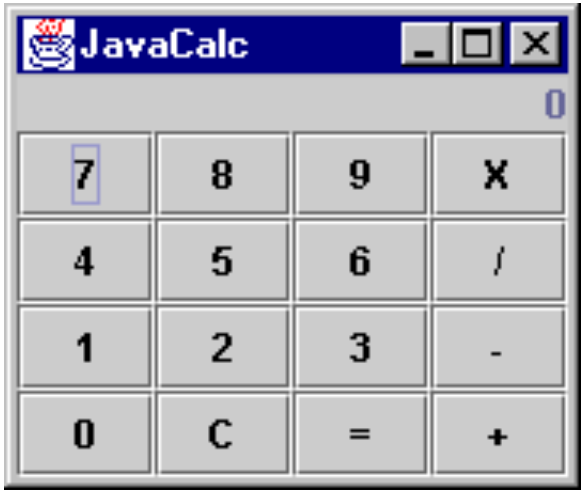




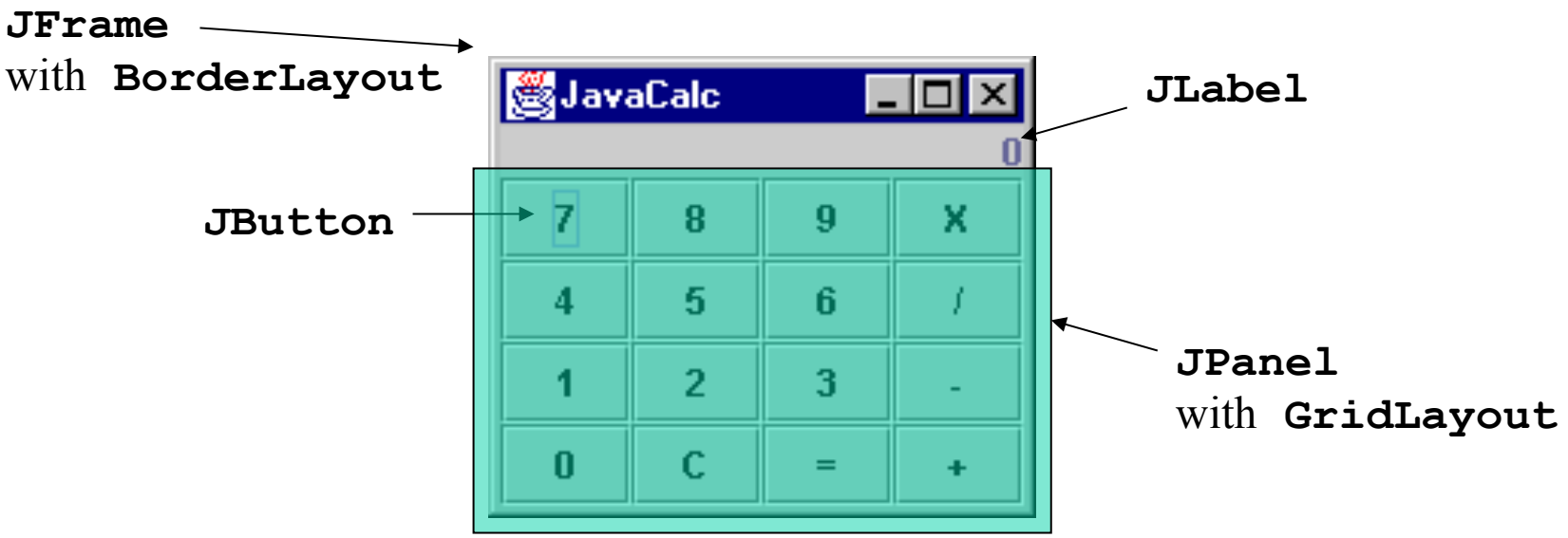
Containers

- A **`JFrame`** is not the only type of container that you can use in Swing
- The subclasses of **`Container`** are:
  - **`JPanel`**
  - **`JWindow`**
  - **`JApplet`**
- **`Window`** is subclassed as follows:
  - **`JDialog`**
  - **`JFrame`**

A Simple 4 Function Calculator



Swing Components



Multithreading

Threads

- Threads are lightweight processes as the overhead of switching between threads is less
- The can be easily spawned
- The Java Virtual Machine spawns a thread when your program is run called the Main Thread

Why do we need threads?

- To enhance parallel processing
- To increase response to the user
- To utilize the idle time of the CPU
- Prioritize your work depending on priority

Example

- Consider a simple web server
- The web server listens for request and serves it
- If the web server was not multithreaded, the requests processing would be in a queue, thus increasing the response time and also might hang the server if there was a bad request.
- By implementing in a multithreaded environment, the web server can serve multiple request simultaneously thus improving response time

Creating threads

- In java threads can be created by extending the Thread class or implementing the Runnable Interface
- It is more preferred to implement the Runnable Interface so that we can extend properties from other classes
- Implement the run() method which is the starting point for thread execution

Running threads

- Example

```
class mythread implements Runnable{
        public void run(){
                System.out.println("Thread Started");
        }
}

class mainclass {
        public static void main(String args[]){
                Thread  t = new Thread(new mythread()); // This is the way to instantiate a
                                                        thread implementing runnable interface
                t.start(); // starts the thread by running the run method
                }
}
```

- Calling t.run() does not start a thread, it is just a simple method call.
- Creating an object does not create a thread, calling start() method creates the thread.

Synchronization

- Synchronization is prevent data corruption
- Synchronization allows only one thread to perform an operation on a object at a time.
- If multiple threads require an access to an object, synchronization helps in maintaining consistency.

Example

```
public class Counter{
    private int count = 0;
    public int getCount(){
        return count;
    }

    public setCount(int count){
        this.count = count;
    }
}
```

- In this example, the counter tells how many an access has been made.
- If a thread is accessing setCount and updating count and another thread is accessing getCount at the same time, there will be inconsistency in the value of count.

Fixing the example

```
public class Counter{
    private static int count = 0;
    public synchronized int getCount(){
            return count;
    }

    public synchoronized setCount(int count){
             this.count = count;
    }
}
```

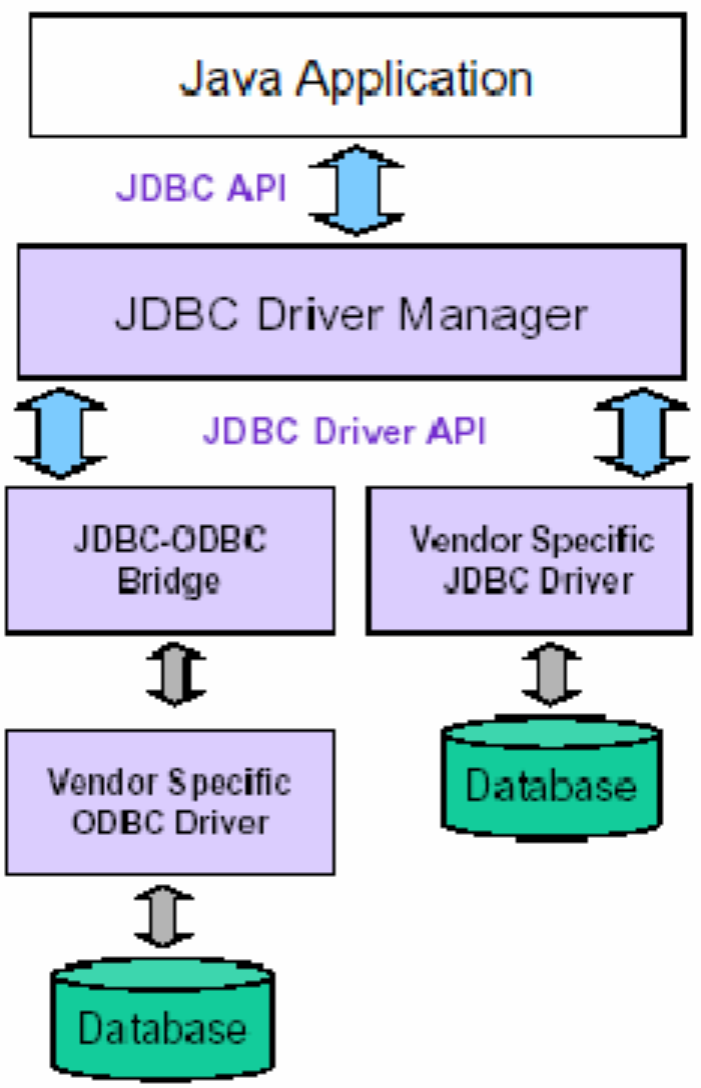
- By adding the synchronized keyword we make sure that when one thread is in the setCount method the other threads are all in waiting state.
- The synchronized keyword places a lock on the object, and hence locks all the other methods which have the keyword synchronized. The lock does not lock the methods without the keyword synchronized and hence they are open to access by other threads.

JDBC

What is JDBC?

- "An API that lets you access virtually any tabular data source from the Java programming language"
    - JDBC Data Access API – JDBC Technology Homepage
  - What's an API?
    - See J2SE documentation
  - What's a tabular data source?
- "… access virtually any data source, from relational databases to spreadsheets and flat files."
  - JDBC Documentation
- We'll focus on accessing Oracle databases

General Architecture



- What design pattern is implied in this architecture?
- What does it buy for us?
- Why is this architecture also multi-tiered?

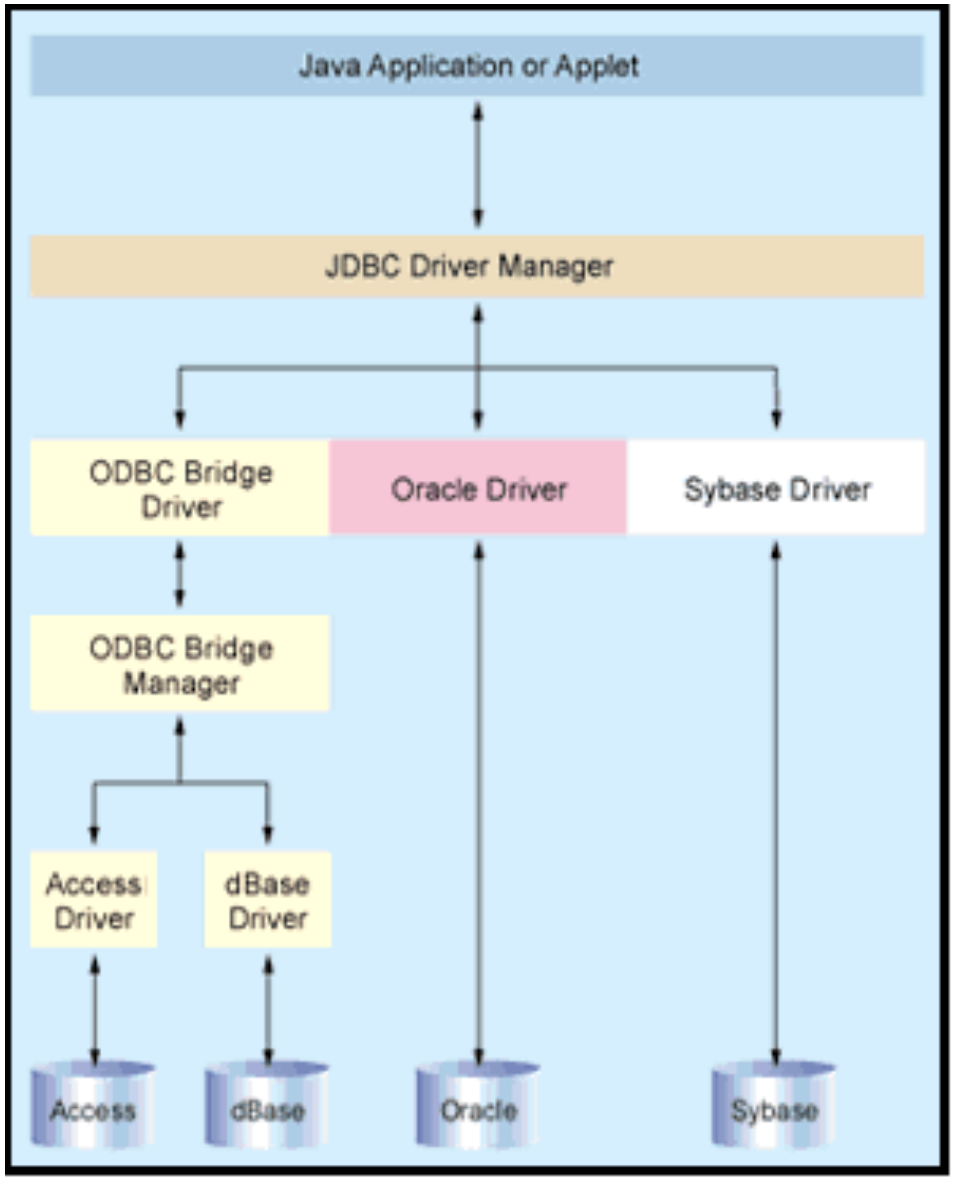


Figure 1. Anatomy of Data Access. The Driver Manager provides a consistent layer between your Java app and back-end database. JDBC works natively (such as with the Oracle driver in this example) or with any ODBC datasource.

Basic steps to use a database in Java

- 1.Establish a **connection**
- 2.Create JDBC **Statements**
- 3.Execute **SQL** Statements
- 4.GET **ResultSet**
- 5.**Close** connections

JAVA Socket Programming

What is a socket?

- Socket
  - The combination of an IP address and a port number. (RFC 793 ,original TCP specification)
  - The name of the Berkeley-derived *application programming interfaces* (APIs) for applications using TCP/IP protocols.
  - Two types
    - Stream socket : reliable two-way connected communication streams
    - Datagram socket

- Socket pair
  - Specified the two end points that uniquely identifies each TCP connection in an internet.
  - 4-tuple: (client IP address, client port number, server IP address, server port number)

Client-server applications

- Implementation of a protocol standard defined in an RFC. (FTP, HTTP, SMTP…)
  - Conform to the rules dictated by the RFC.
  - Should use the port number associated with the protocol.

- Proprietary client-server application.
  - A single developer(or team) creates both client and server program.
  - The developer has complete control.
  - Must be careful not to use one of the well-known port number defined in the RFCs.

  * well-known port number : managed by the Internet Assigned Numbers Authority(IANA)

Socket Programming with TCP

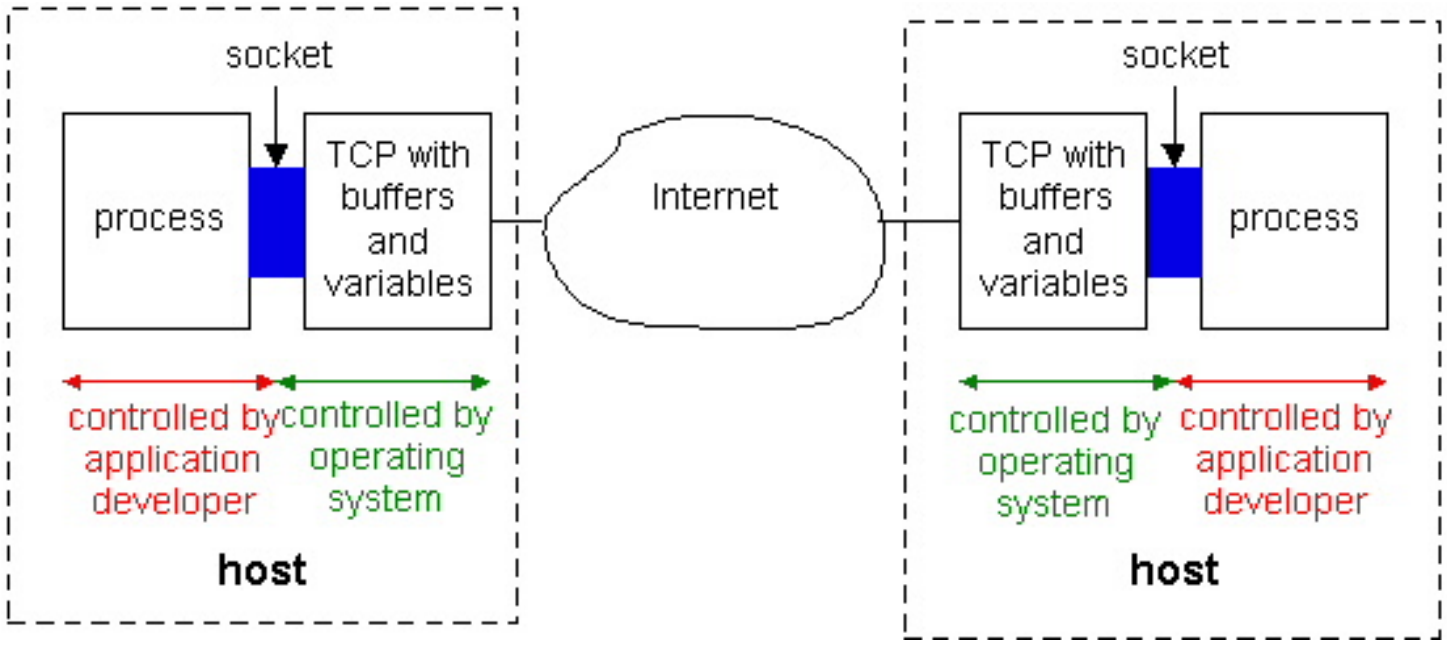


Figure 2.6-1: Processes communicating through TCP sockets

The application developer has the ability to fix a few TCP parameters, such as maximum buffer and maximum segment sizes.

Sockets for server and client

- Server
  - Welcoming socket
    - Welcomes some initial contact from a client.
  - Connection socket
    - Is created at initial contact of client.
    - New socket that is dedicated to the particular client.

- Client
  - Client socket
    - Initiate a TCP connection to the server by creating a socket object. (Three-way handshake)
    - Specify the address of the server process, namely, the IP address of the server and the port number of the process.

Socket functional calls

- socket (): Create a socket
- bind(): bind a socket to a local IP address and port #
- listen(): passively waiting for connections
- connect(): initiating connection to another socket
- accept(): accept a new connection
- Write(): write data to a socket
- Read(): read data from a socket
- sendto(): send a datagram to another UDP socket
- recvfrom(): read a datagram from a UDP socket
- close(): close a socket (tear down the connection)

Sockets

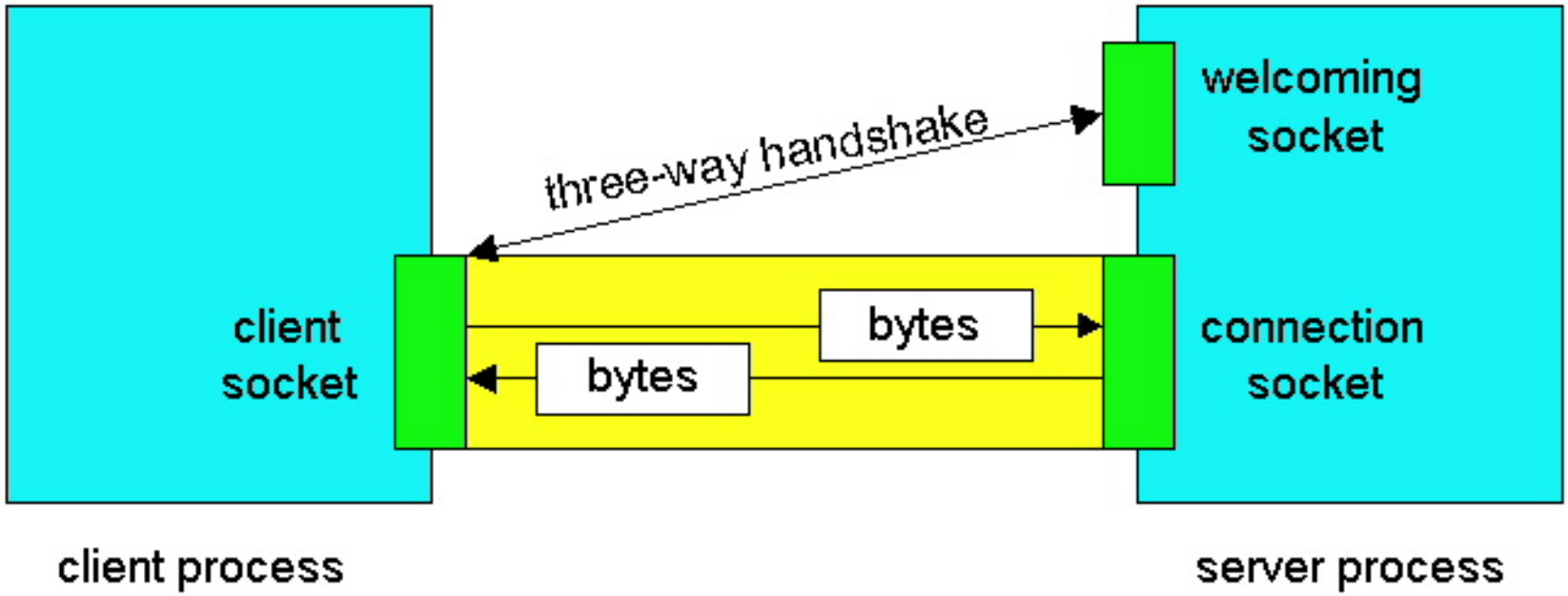
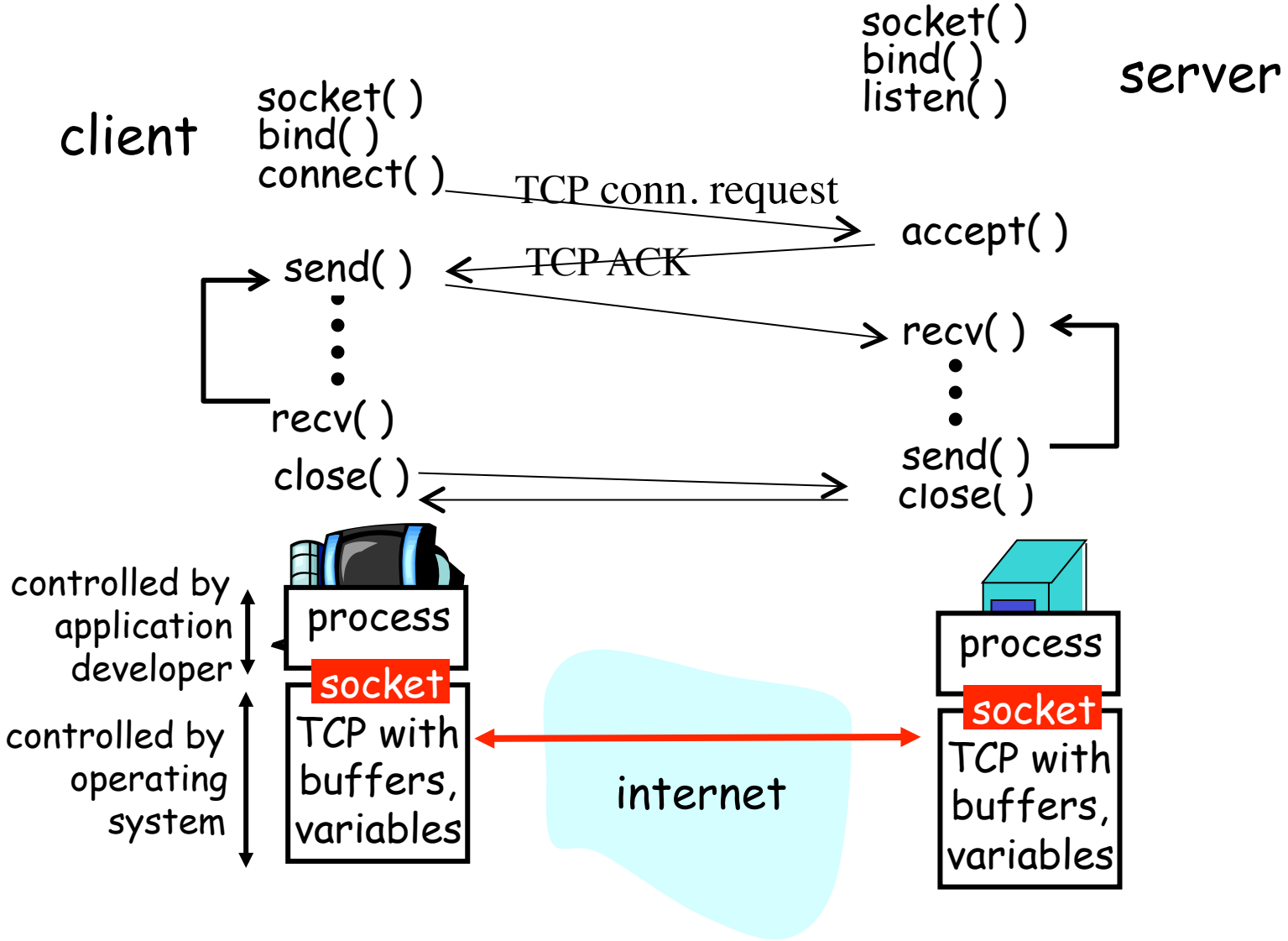


Figure 2.6-2: Client socket, welcoming socket and connection socket

Socket-programming using TCP

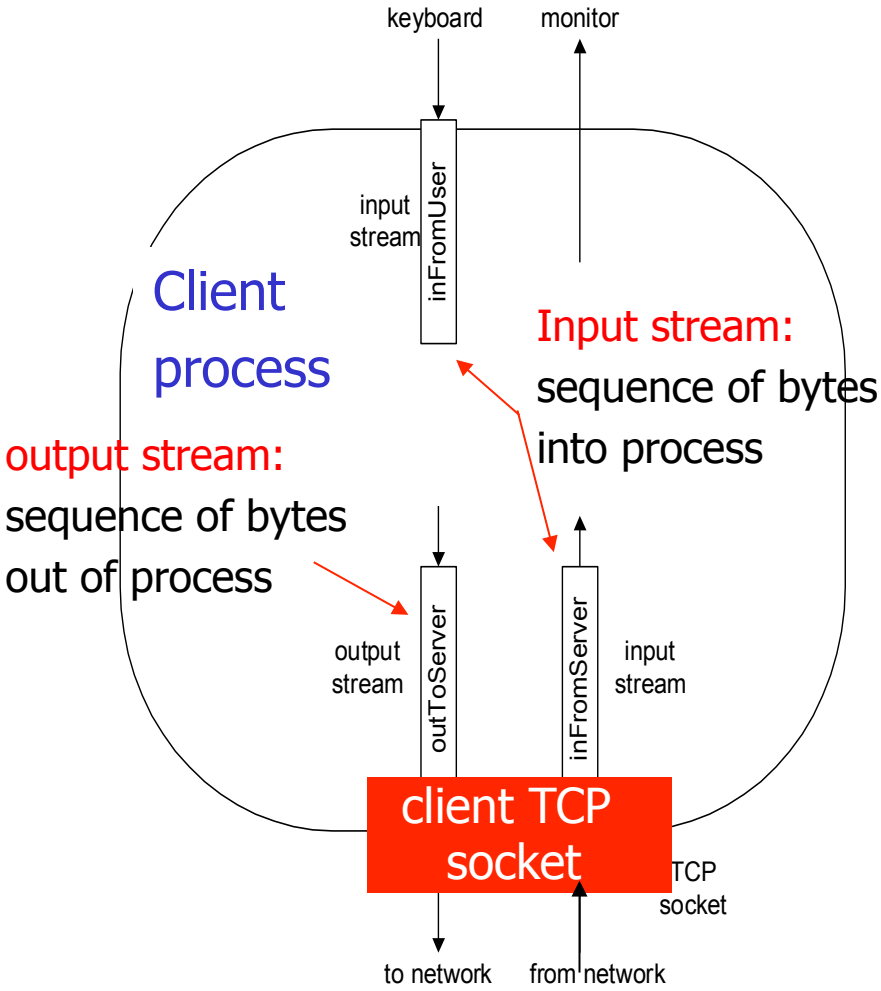
TCP service: reliable byte stream transfer



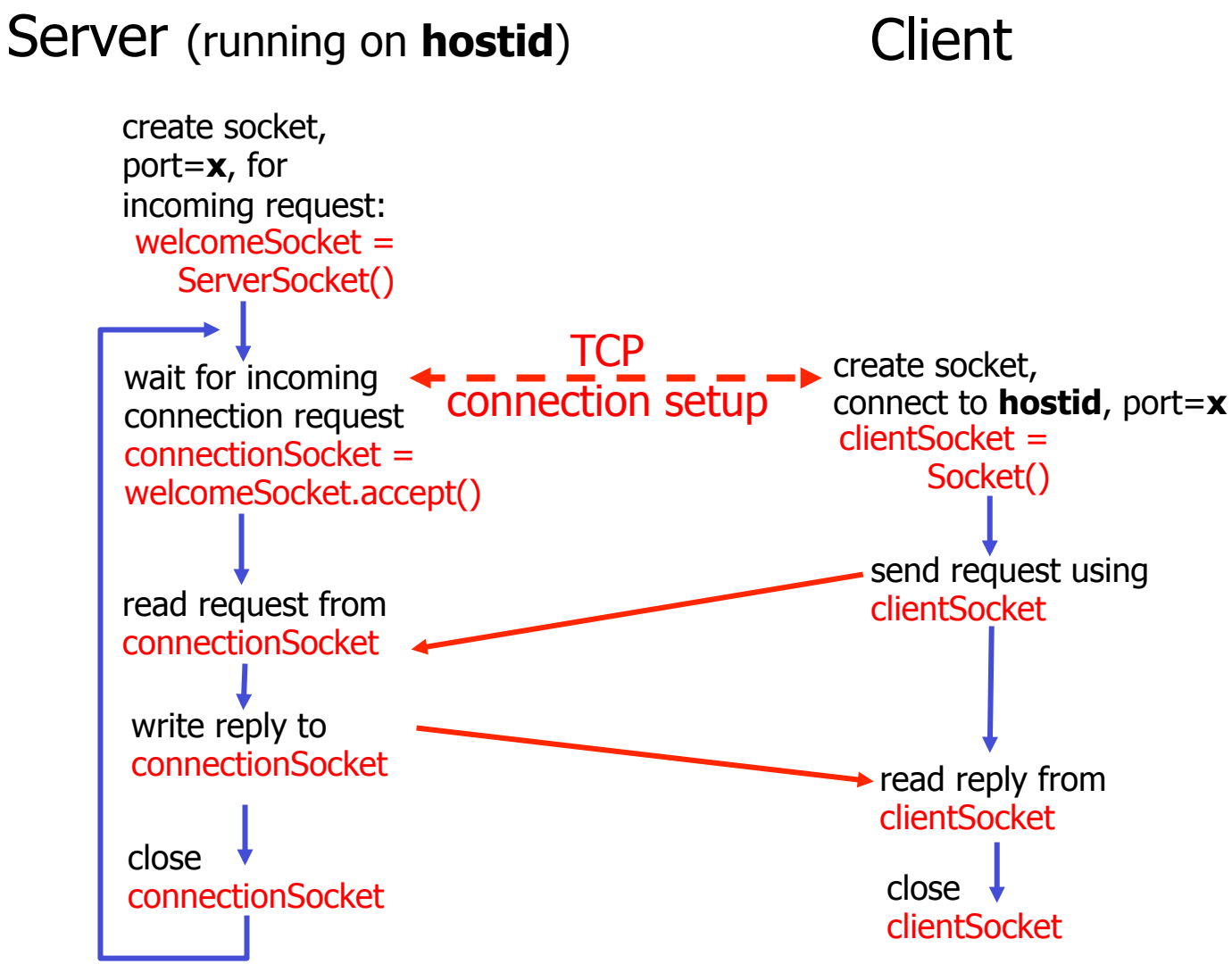
Socket programming with TCP

Example client-server app:

- client reads line from standard input (**inFromUser** stream) , sends to server via socket (**outToServer** stream)
- server reads line from socket
- server converts line to uppercase, sends back to client
- client reads, prints modified line from socket (**inFromServer** stream)



Client/server socket interaction: TCP



JAVA TCP Sockets

- In Package java.net
  - java.net.Socket
    - Implements client sockets (also called just "sockets").
    - An endpoint for communication between two machines.
    - Constructor and Methods
      - Socket(String host, int port): Creates a stream socket and connects it to the specified port number on the named host.
      - InputStream getInputStream()
      - OutputStream getOutputStream()
      - close()
  - java.net.ServerSocket
    - Implements server sockets.
    - Waits for requests to come in over the network.
    - Performs some operation based on the request.
    - Constructor and Methods
      - ServerSocket(int port)
      - Socket Accept(): Listens for a connection to be made to this socket and accepts it. This method blocks until a connection is made.

Socket Programming with UDP

- UDP
  - Connectionless and unreliable service.
  - There isn't an initial handshaking phase.
  - Doesn't have a pipe.
  - transmitted data may be received out of order, or lost

- Socket Programming with UDP
  - No need for a welcoming socket.
  - No streams are attached to the sockets.
  - the sending hosts creates "packets" by attaching the IP destination address and port number to each batch of bytes.
  - The receiving process must unravel to received packet to obtain the packet's information bytes.

Client/server socket interaction: UDP