Chapter 11: Multithreading

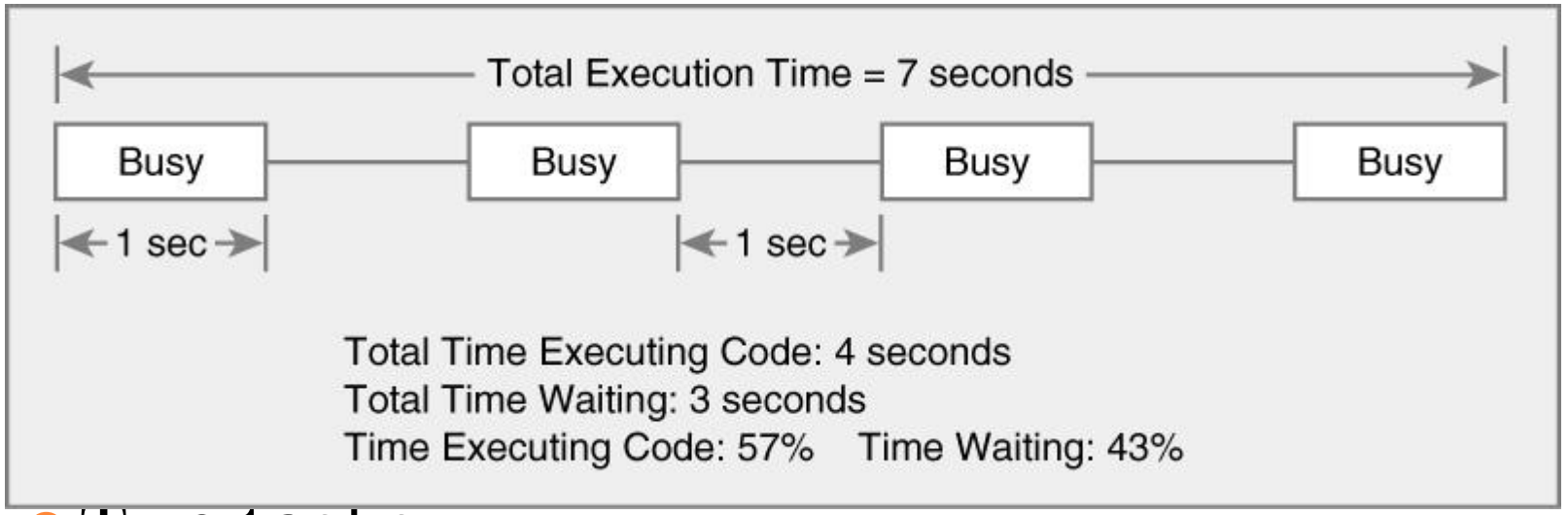
- Multiple tasks for computer
  - Draw & display images on screen
  - Check keyboard & mouse input
  - Send & receive data on network
  - Read & write files to disk
  - Perform useful computation (editor, browser, game)
- How does computer do everything at once?
  - Multitasking
  - Multiprocessing

Multitasking (Time-Sharing)

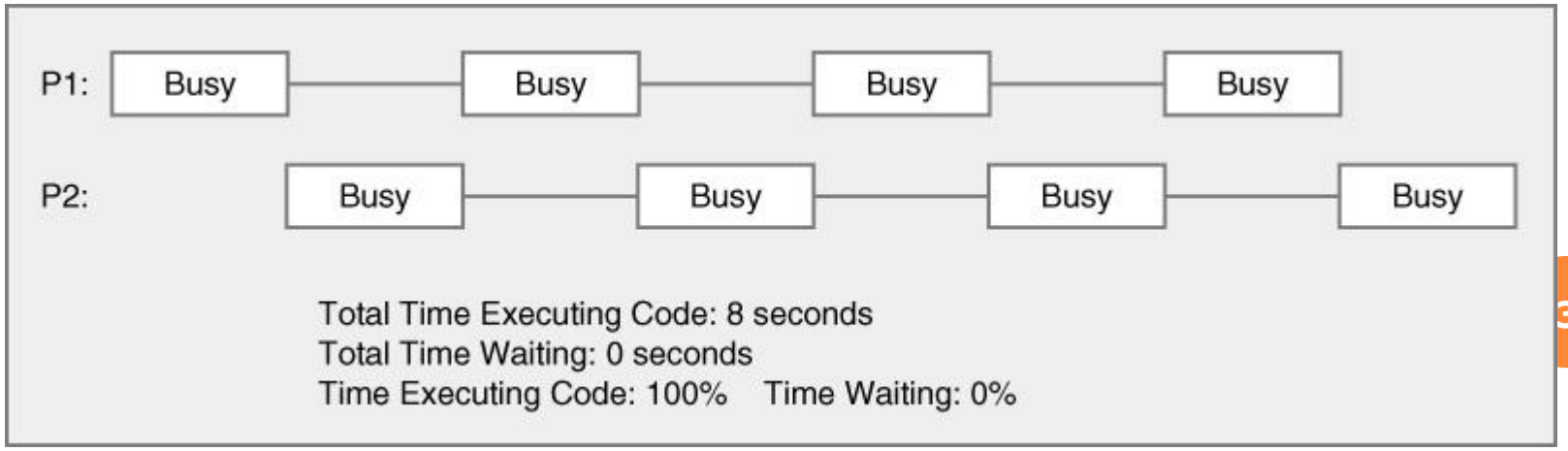
- Approach
  - Computer does some work on a task
  - Computer then quickly switch to next task
  - Tasks managed by operating system (scheduler)
- Computer seems to work on tasks concurrently
- Can improve performance by reducing waiting

Multitasking Can Aid Performance

- Single task



- Two tasks



Multiprocessing (Multithreading)

- Approach
  - Multiple processing units (multiprocessor)
  - Computer works on several tasks in parallel
  - Performance can be improved

Dual-core AMD Athlon X2

32 processor Pentium Xeon

4096 processor Cray X1

Perform Multiple Tasks Using…

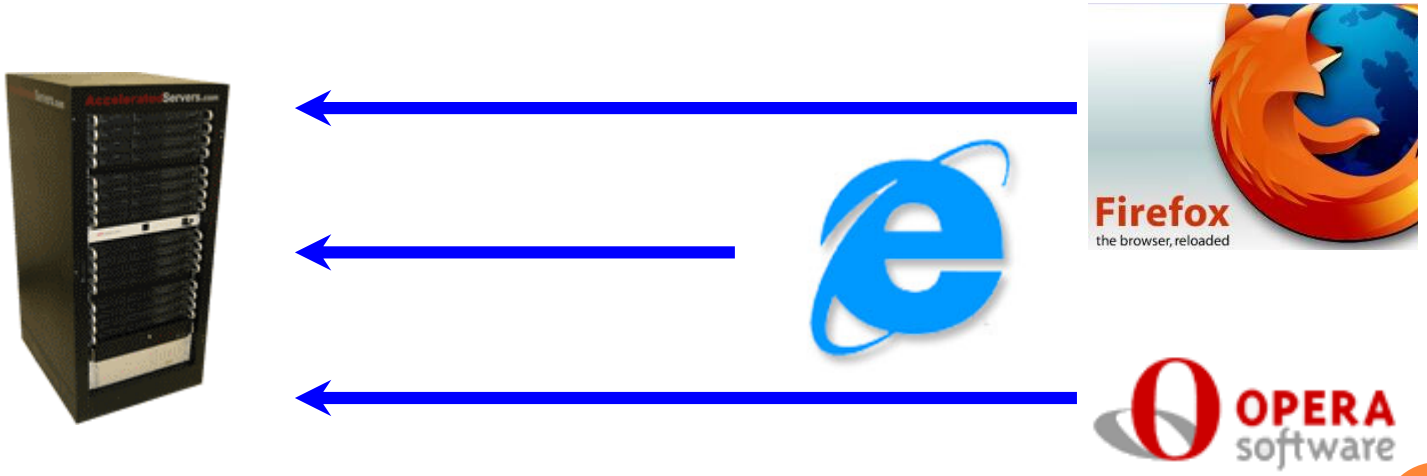
- Process
  - Definition – executable program loaded in memory
  - Has own address space
    - Variables & data structures (in memory)
  - Each process may execute a different program
  - Communicate via operating system, files, network
  - May contain multiple threads

Perform Multiple Tasks Using…

- Thread
  - Definition – sequentially executed stream of instructions
  - Shares address space with other threads
  - Has own execution context
    - Program counter, call stack (local variables)
  - Communicate via shared access to data
  - Multiple threads in process execute same program
  - Also known as “lightweight process”

Motivation for Multithreading

- Captures logical structure of problem
  - May have concurrent interacting components
  - Can handle each component using separate thread
  - Simplifies programming for problem
- Example



Web Server uses threads to handle …

Multiple simultaneous web browser requests

Motivation for Multithreading

- Better utilize hardware resources
  - When a thread is delayed, compute other threads
  - Given extra hardware, compute threads in parallel
  - Reduce overall execution time
- Example

Multiple simultaneous web browser requests…

Handled faster by multiple web servers

Multithreading Overview

- Motivation & background
- Threads
  - Creating Java threads
  - Thread states
  - Scheduling
- Synchronization
  - Data races
  - Locks
  - Wait / Notify

Programming with Threads

- Concurrent programming
  - Writing programs divided into independent tasks
  - Tasks may be executed in parallel on multiprocessors
- Multithreading
  - Executing program with multiple threads in parallel
  - Special form of multiprocessing

Creating Threads in Java

- You have to specify the work you want the thread to do
- Define a class that implements the Runnable interface

```
public interface Runnable {
    public void run();
}
```

- Put the work in the run method
- Create an instance of the worker class and create a thread to run it
  - or hand the worker instance to an executor

Thread Class

```
public class Thread {

    public Thread(Runnable R);    // Thread ⇒ R.run()
    public Thread(Runnable R, String name);

    public void start();          // begin thread execution
    …
}
```

More Thread Class Methods

```
public class Thread {
    …
    public String getName();
    public void interrupt();
    public boolean isAlive();
    public void join();
    public void setDaemon(boolean on);
    public void setName(String name);
    public void setPriority(int level);

    public static Thread currentThread();

    public static void sleep(long milliseconds);
    public static void yield();
}
```

Creating Threads in Java

- Runnable interface
  - Create object implementing Runnable interface
  - Pass it to Thread object via Thread constructor
- Example

```
public class MyT implements Runnable {
    public void run() {
        …   // work for thread
    }
}
Thread t = new Thread(new MyT());  // create thread
t.start();                                  // begin running
    thread
…                                  // thread executing in parallel
```

Alternative (Not Recommended)

- Directly extend Thread class

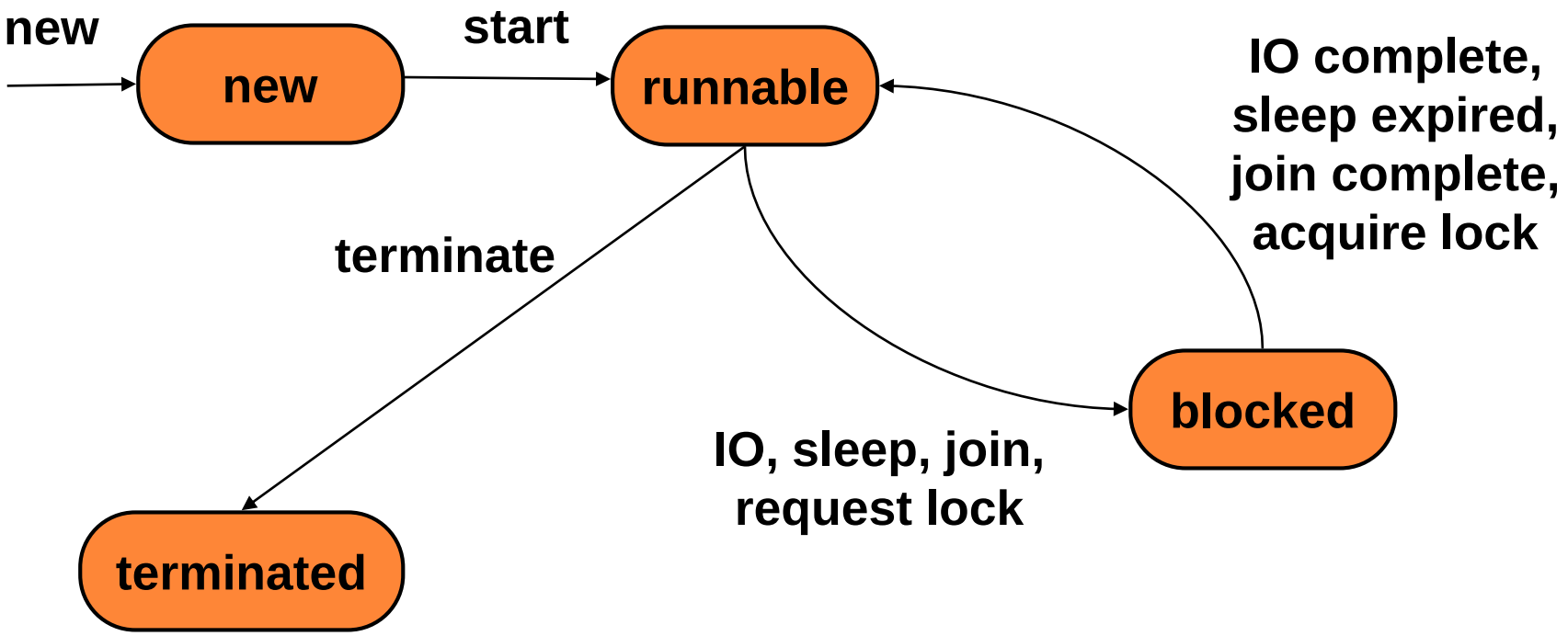
```
public class MyT extends Thread {
public void run() {
    …   // work for thread
}
}
MyT t = new MyT();          // create thread
t.start();                  // begin running thread
```

Threads – Thread States

- Java thread can be in one of these states
  - New – thread allocated & waiting for start()
  - Runnable – thread can execute
  - Blocked – thread waiting for event (I/O, etc.)
  - Terminated – thread finished
- Transitions between states caused by
  - Invoking methods in class Thread
    - start(), yield(), sleep()
  - Other (external) events
    - Scheduler, I/O, returning from run()…

Threads – Thread States

State diagram



Threads – Scheduling

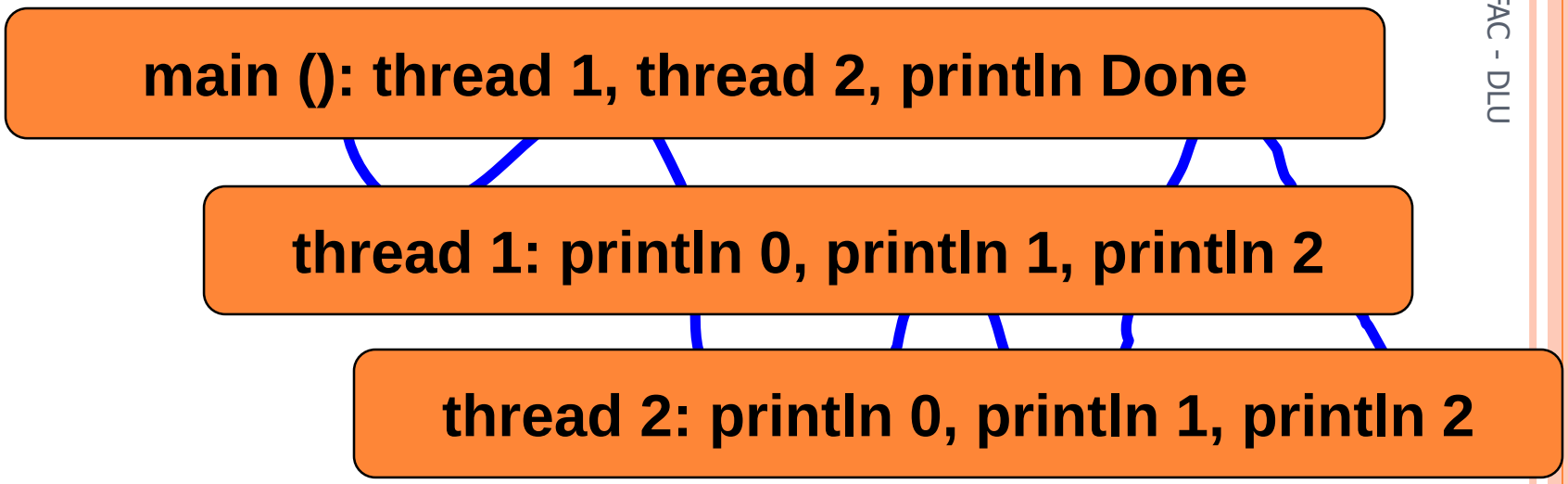
- Scheduler
  - Determines which runnable threads to run
  - Can be based on thread priority
  - Part of OS or Java Virtual Machine (JVM)
  - Many computers can run multiple threads simultaneously (or nearly so)

Java Thread Example

```
public class ThreadExample implements Runnable {
    public void run() {
        for (int i = 0; i < 3; i++)
            System.out.println(i);

    }
    public static void main(String[] args) {
      new Thread(new ThreadExample()).start();
      new Thread( new ThreadExample()).start();
      System.out.println("Done");
    }
}
```

Java Thread Example – Output

- Possible outputs
  - 0,1,2,0,1,2,Done // thread 1, thread 2, main()
  - 0,1,2,Done,0,1,2 // thread 1, main(), thread 2
  - Done,0,1,2,0,1,2 // main(), thread 1, thread 2
  - 0,0,1,1,2,Done,2 // main() & threads interleaved



Might not see different interleavings

- The threads in that example are too short
- Each started thread will probably complete before the next thread starts

- Let's make more threads that run longer

Data Races

```
public class DataRace implements Runnable {
 static volatile int x;
 public void run() {
    for (int i = 0; i < 10000; i++) {
       x++;
       x--;
    }
}
 public static void main(String[] args) throws Exception {
     Thread [] threads = new Thread[100];
     for (int i = 0; i < threads.length; i++)
        threads[i] = new Thread(new DataRace());
     for (int i = 0; i < threads.length; i++)
        threads[i].start();
     for (int i = 0; i < threads.length; i++)
        threads[i].join();
     System.out.println(x);    // x not always 0!
```

USING SYNCHRONIZATION

```
public class DataRace implements Runnable {
 static volatile int x;
 static Object lock = new Object();
 public void run() {
    for (int i = 0; i < 10000; i++)
       synchronized(lock) {
          x++;  x--;
    }
 }
 public static void main(String[] args) throws Exception {
     Thread [] threads = new Thread[100];
     for (int i = 0; i < threads.length; i++)
        threads[i] = new Thread(new DataRace());
     for (int i = 0; i < threads.length; i++)
        threads[i].start();
     for (int i = 0; i < threads.length; i++)
        threads[i].join();
     System.out.println(x);    // x always 0!
 }
}
```

Chapter 12: Multimedia

- **Loading, Displaying and Scaling Images**
- **Loading and Playing Audio Clips**
- **Animating a Series of Images**
- **Customizing Applets via the HTML param Tag**

Loading, Displaying and Scaling Images

- Java Multimedia
  - Graphics, images, animations, sounds, and video
    - Begin with images
- Class **`Image`** (**`java.awt`**)
  - Abstract class, cannot create an object directly
    - Must request that an **`Image`** be loaded and returned to you
  - Class **`Applet`** (superclass of **`JApplet`**) has this method
    - **`getImage( imageLocation, filename );`**
    - **`imageLocation - getDocumentBase()`** - URL (address) of HTML file
    - **`filename`** - Java supports **`.gif`** and **`.jpg`** (**`.jpeg`**)

Loading, Displaying and Scaling Images (II)

- Displaying **Image**s with **drawImage**
  - Many overloaded versions

```
g.drawImage( myImage, x, y, ImageObserver );
```

    - **myImage** – **Image** object
    - **x,y** - coordinates to display image
    - **ImageObserver** - object on which image is displayed
      - Use "**this**" to indicate the applet
      - Can be any object that implements **ImageObserver** interface

  -

```
g.drawImage( myImage, x, y, width, height,
             ImageObserver );
```

    - **width** and **height** - dimensions of image (automatically scaled)
      - **getWidth()**, **getHeight()** - return dimensions of applet

Loading, Displaying and Scaling Images (III)

- Class **ImageIcon**
  - Not an abstract class (can create objects)
  - Example constructor

```
private ImageIcon myIcon;
myIcon = new ImageIcon( "myIcon.gif" );
```

- Displaying **Icons** with method **paintIcon**

```
myIcon.paintIcon( Component, Graphics, x, y )
```

  - **Component** - **Component** object on which to display image (**this**)
  - **Graphics** - **Graphics** object used to render image (**g**)
  - **x**, **y** - coordinates of **Icon**

Loading, Displaying and Scaling Images (IV)

- Usage
  - **`ImageIcon`**s are simpler than **`Image`**s
    - Create objects directly
    - No need for **`ImageObserver`** reference
  - However, cannot scale **`ImageIcon`**s
- Scaling
  - Use **`ImageIcon`** method **`getImage`**
    - Returns **`Image`** reference
    - This can be used with **`drawImage`** and be scaled

```java
// Fig. 30.1: LoadImageAndScale.java
// Load an image and display it in its original size
// and scale it to twice its original width and height.
// Load and display the same image as an ImageIcon.
import java.applet.Applet;
import java.awt.*;
import javax.swing.*;

public class LoadImageAndScale extends JApplet {
   private Image logo1;
   private ImageIcon logo2;

   // load the image when the applet is loaded
   public void init()
   {
      logo1 = getImage( getDocumentBase(), "logo.gif" );
      logo2 = new ImageIcon( "logo.gif" );
   }

   // display the image
   public void paint( Graphics g )
   {
      // draw the original image
      g.drawImage( logo1, 0, 0, this );

      // draw the image scaled to fit the width of the applet
      // and the height of the applet minus 120 pixels
      g.drawImage( logo1, 0, 120,
                   getWidth(), getHeight() - 120, this );

      // draw the icon using its paintIcon method
```

Loading and Playing Audio Clips

- Audio clips
  - Require speakers and a sound board
  - Sound engine - plays audio clips
    - Supports **.au**, **.wav**, .**aif**, **.mid**
    - Java Media Framework supports additional formats
- Playing audio clips
  - **play** method in **Applet**
  - Plays clip once, marked for garbage collection when finished

```
play( location, soundFileName );
```

location - getDocumentBase (URL of HTML file)

```
play( soundURL );
```

soundURL - URL that contains location and filename of clip

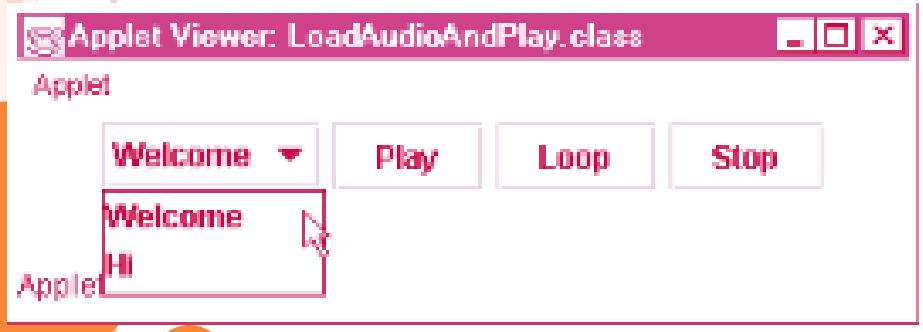
Loading and Playing Audio Clips (II)

- Playing audio clips
  - Method **`play`** from **`AudioClip`** interface
  - More flexible than **`Applet`** method **`play`**
    - Audio stored in program, can be reused
  - **`getAudioClip`**
    - Returns reference to an **`AudioClip`**
    - Same format as **`Applet`** method **`play`**
      - `getAudioClip( location, filename )`
      - `getAudioClip( soundURL )`
  - Once **`AudioClip`** loaded, use methods
    - **`play`** - plays audio once
    - **`loop`** - continuous loops audio in background
    - **`stop`** - terminates clip that is currently playing

```
1  // Fig. 30.2: LoadAudioAndPlay.java
2  // Load an audio clip and play it.
3  import java.applet.*;
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class LoadAudioAndPlay extends JApplet {
9     private AudioClip sound1, sound2, currentSound;
10    private JButton playSound, loopSound, stopSound;
11    private JComboBox chooseSound;
12
13    // load the image when the applet begins executing
14    public void init()
15    {
16       Container c = getContentPane();
17       c.setLayout( new FlowLayout() );
18
19       String choices[] = { "Welcome", "Hi" };
20       chooseSound = new JComboBox( choices );
21       chooseSound.addItemListener(
22          new ItemListener() {
23             public void itemStateChanged( ItemEvent e )
24             {
25                currentSound.stop();
26
27                currentSound =
28                   chooseSound.getSelectedIndex() == 0 ?
29                      sound1 : sound2;
30             }
31          }
32       );
```

```
33          c.add( chooseSound );
34
35          ButtonHandler handler = new ButtonHandler();
36          playSound = new JButton( "Play" );
37          playSound.addActionListener( handler );
38          c.add( playSound );
39          loopSound = new JButton( "Loop" );
40          loopSound.addActionListener( handler );
41          c.add( loopSound );
42          stopSound = new JButton( "Stop" );
43          stopSound.addActionListener( handler );
44          c.add( stopSound );
45
46          sound1 = getAudioClip(
47                      getDocumentBase(), "welcome.wav" );
48          sound2 = getAudioClip(
49                      getDocumentBase(), "hi.au" );
50          currentSound = sound1;
51       }
52
53       // stop the sound when the user switches Web pages
54       // (i.e., be polite to the user)
55       public void stop()
56       {
57          currentSound.stop();
58       }
59
60       private class ButtonHandler implements ActionListener {
61          public void actionPerformed( ActionEvent e )
62          {
63             if ( e.getSource() == playSound )
64                currentSound.play();
```

```
65          else if ( e.getSource() == loopSound )
66             currentSound.loop();
67          else if ( e.getSource() == stopSound )
68             currentSound.stop();
69       }
70    }
71 }
```



Animating a Series of Images

- Following example
  - Use a series of images stored in an array
  - Use same techniques to load and display **`ImageIcon`**s
- Class **`Timer`**
  - Generates **`ActionEvents`** at a fixed interval in milliseconds

    `Timer ( animationDelay, ActionListener );`

    `ActionListener` - **`ActionListener`** that will respond to **`ActionEvents`**
  - Methods
    - **`start`**
    - **`stop`**
    - **`restart`**
    - **`isRunning`**

Animating a Series of Images (II)

- Method **`repaint`**
  - Calls **`update`**, which calls **`paintComponent`**
    - Subclasses of **`JComponent`** should draw in method **`paintComponent`**
    - Call superclass's **`paintComponent`** to make sure Swing components displayed properly
- View area
  - Width and height specify entire window, not client area
  - **`Dimension`** objects
    - Contain **`width`** and **`height`** values

```
myDimObject = new Dimension( 100, 200 );
myDimObject.width
```

Animating a Series of Images (III)

- **`getImageLoadStatus`**
  - **`ImageIcon`** method
    - Determines if image is completely loaded into memory
    - Only complete images should be displayed (smooth animation)
  - If loaded, returns **`MediaTracker.COMPLETE`**
  - **`MediaTracker`**
    - Can determine when images are loaded, or force program to wait if not
    - **`ImageIcon`** creates our **`MediaTracker`** for us

```java
// Fig. 30.3: LogoAnimator.java
// Animation a series of images
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class LogoAnimator extends JPanel
                          implements ActionListener {
   protected ImageIcon images[];
   protected int totalImages = 30,
                 currentImage = 0,
                 animationDelay = 50; // 50 millisecond delay
   protected Timer animationTimer;

   public LogoAnimator()
   {
      setSize( getPreferredSize() );

      images = new ImageIcon[ totalImages ];

      for ( int i = 0; i < images.length; ++i )
         images[ i ] =
            new ImageIcon( "images/deitel" + i + ".gif" );

      startAnimation();
   }

   public void paintComponent( Graphics g )
   {
      super.paintComponent( g );

```

```
32       if ( images[ currentImage ].getImageLoadStatus() ==
33            MediaTracker.COMPLETE ) {
34          images[ currentImage ].paintIcon( this, g, 0, 0 );
35          currentImage = ( currentImage + 1 ) % totalImages;
36       }
37    }
38
39    public void actionPerformed( ActionEvent e )
40    {
41       repaint();
42    }
43
44    public void startAnimation()
45    {
46       if ( animationTimer == null ) {
47          currentImage = 0;
48          animationTimer = new Timer( animationDelay, this );
49          animationTimer.start();
50       }
51       else  // continue from last image displayed
52          if ( ! animationTimer.isRunning() )
53             animationTimer.restart();
54    }
55
56    public void stopAnimation()
57    {
58       animationTimer.stop();
59    }
60
61    public Dimension getMinimumSize()
62    {
63       return getPreferredSize();
64    }
```

```
65
66     public Dimension getPreferredSize()
67     {
68        return new Dimension( 160, 80 );
69     }
70
71     public static void main( String args[] )
72     {
73        LogoAnimator anim = new LogoAnimator();
74
75        JFrame app = new JFrame( "Animator test" );
76        app.getContentPane().add( anim, fBorderLayout.CENTER );
77
78        app.addWindowListener(
79           new WindowAdapter() {
80              public void windowClosing( WindowEvent e )
81              {
82                 System.exit( 0 );
83              }
84           }
85        );
86
87        // The constants 10 and 30 are used below to size the
88        // window 10 pixels wider than the animation and
89        // 30 pixels taller than the animation.
90        app.setSize( anim.getPreferredSize().width + 10,
91                     anim.getPreferredSize().height + 30 );
92        app.show();
93     }
94  }
```

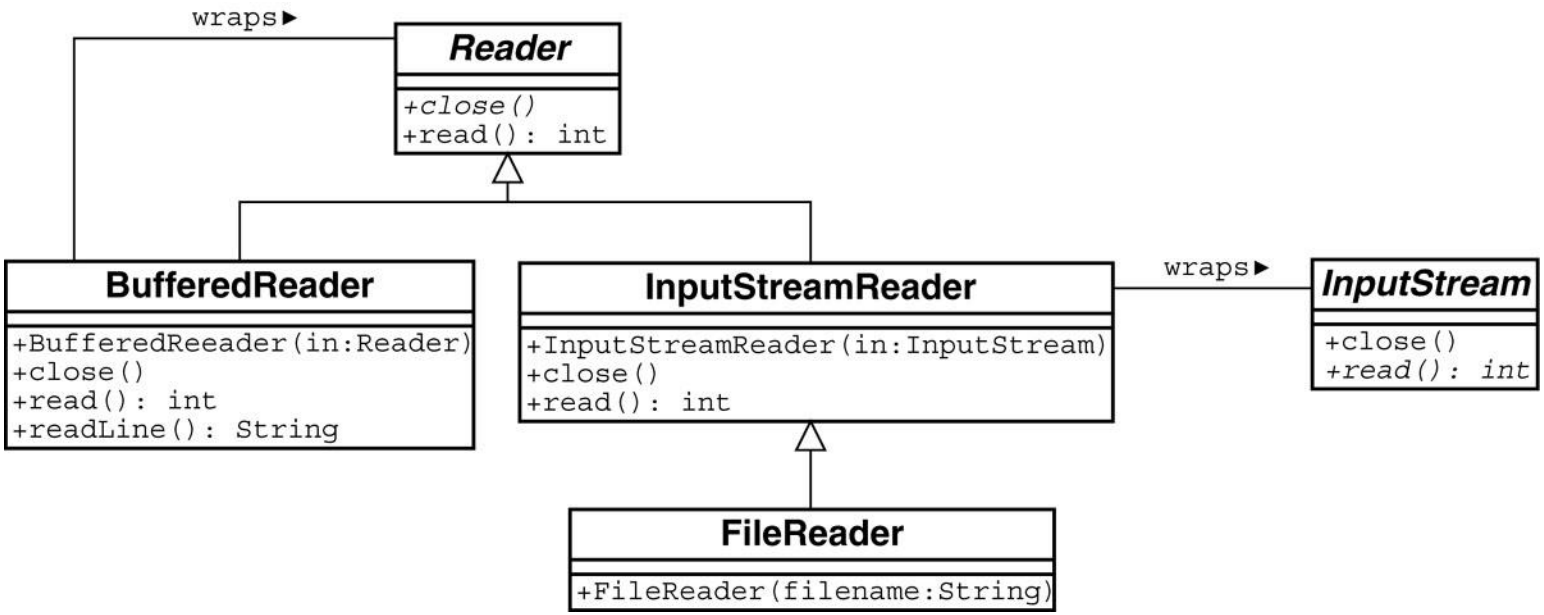
Program Output

Chapter 13
Input and Output

In this chapter we will:

- Read a File
- Write a File

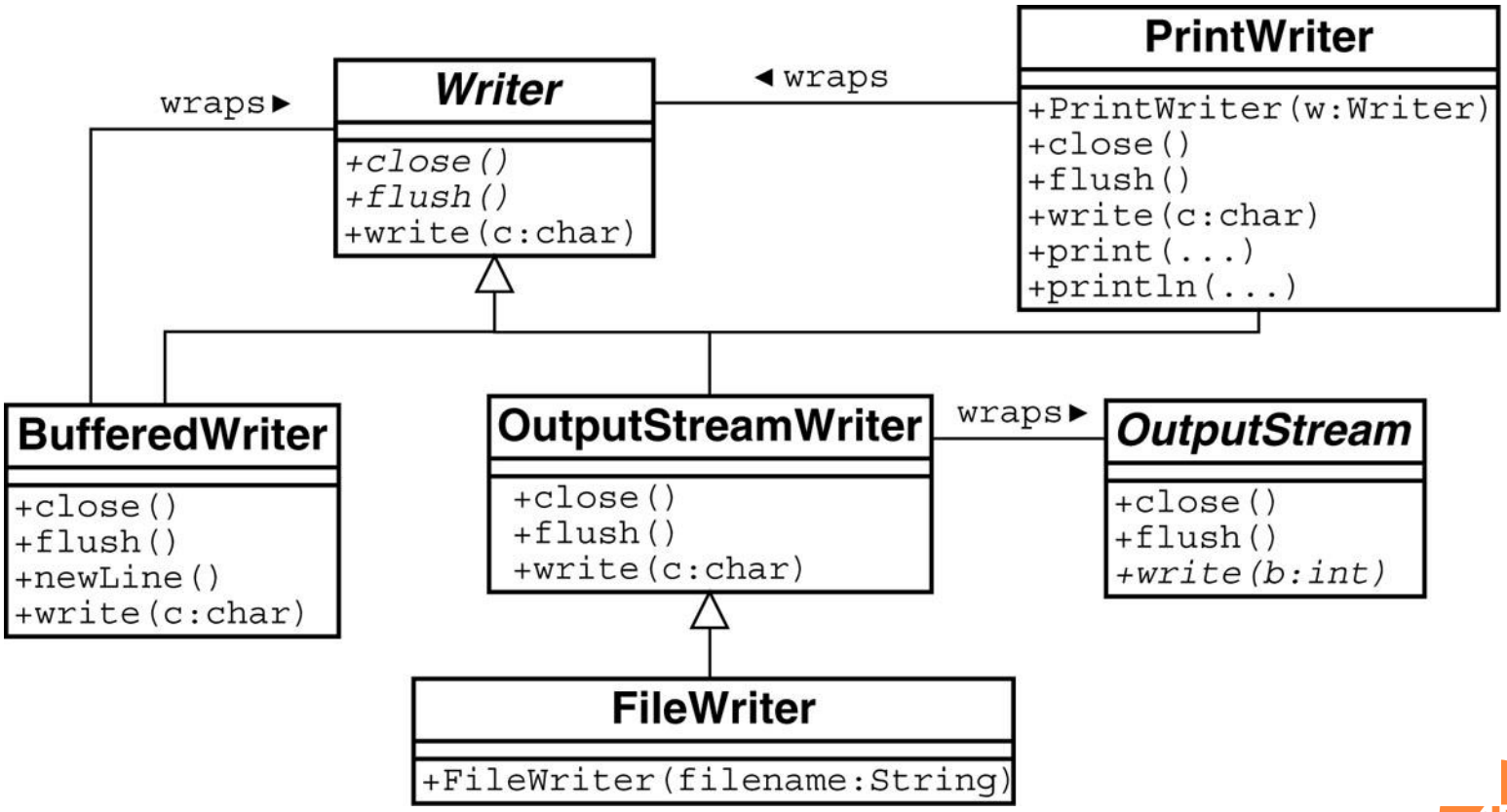
File Input


wraps►
Reader
+close()
+read(): int
BufferedReader
+BufferedReeader(in:Reader)
+close()
+read(): int
+readLine(): String
InputStreamReader
+InputStreamReader(in:InputStream)
+close()
+read(): int
wraps►
InputStream
+close()
+read(): int
FileReader
+FileReader(filename:String)

Example
Read File

```
public static void ReadFile(String filename)
        {
                FileReader fr;
                try{
                        fr = new FileReader(filename);
                        char str[]={2};
                        try{
                                while(fr.read(str)!=-1)
                                {
                                        System.out.print(str);
                                }
                        }
                        catch(IOException ioe)
                        {}
                }
                catch(Exception e)
                {
                        System.out.print(e.getMessage());
                }
        }
```

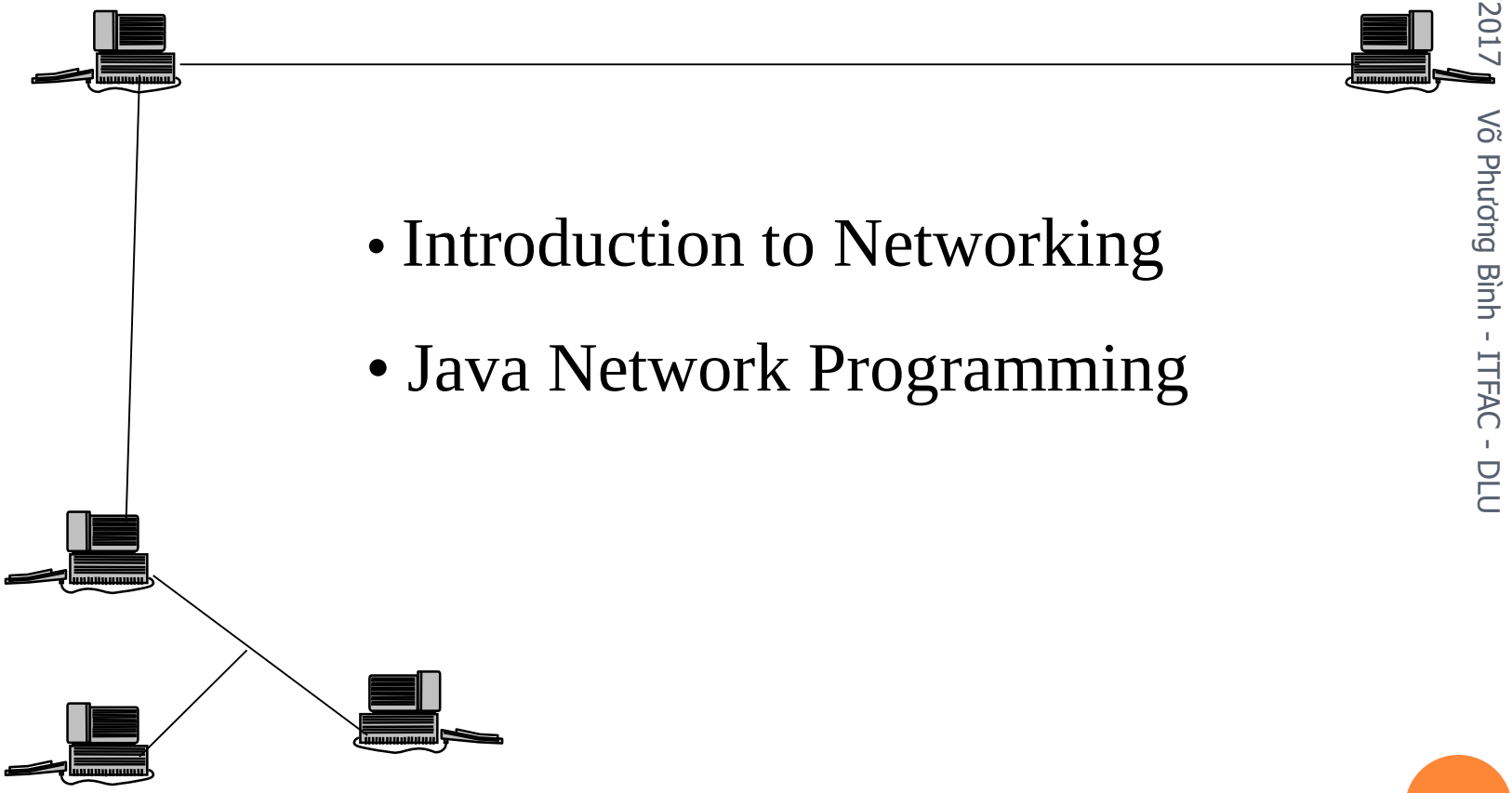
File Output


Writer
+close()
+flush()
+write(c:char)
wraps ▶
◀ wraps
PrintWriter
+PrintWriter(w:Writer)
+close()
+flush()
+write(c:char)
+print(...)
+println(...)
BufferedWriter
+close()
+flush()
+newLine()
+write(c:char)
OutputStreamWriter
+close()
+flush()
+write(c:char)
wraps ▶
OutputStream
+close()
+flush()
+write(b:int)
FileWriter
+FileWriter(filename:String)

Example
Write to File

```
public static void WriteFile(String filename, String str)
        {
                FileWriter fw;
                try{
                        fw = new FileWriter(filename, true);
                        fw.write(str);
                        fw.close();
                }
                catch(Exception e)
                {
                        System.out.print(e.getMessage());
                }
        }
```

Chapter 14: Networking



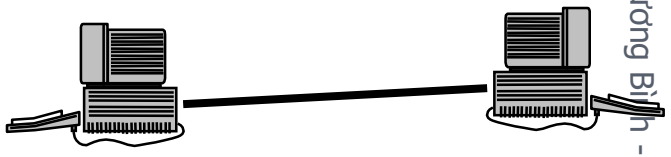
- Introduction to Networking
- Java Network Programming

Networking with Java

- Java supports stream sockets and datagram sockets

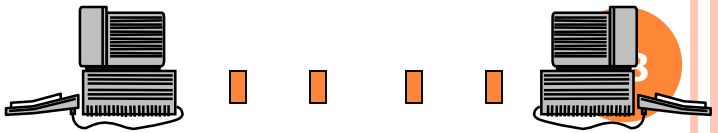
Stream socket:

- Data flows between the processes while the connection is in place
- Connection-oriented
- The protocol for transmission is TCP



Datagram socket

- individual packets of information are transmitted
- Connection-less service
- The protocol for transmission is UDP



Networking with Java

Uses for TCP Sockets

- Network overhead is not important
- Reliable transfer in required

Uses for UDP Sockets

- Network overhead must be minimized
- Data reliability is not crucial
- small independent applications packets needed

Networking with Java

Relevant Classes for network programming with Java

Socket - a TCP connection socket

ServerSocket - a TCP server socket

DatagramPacket - a UDP packet

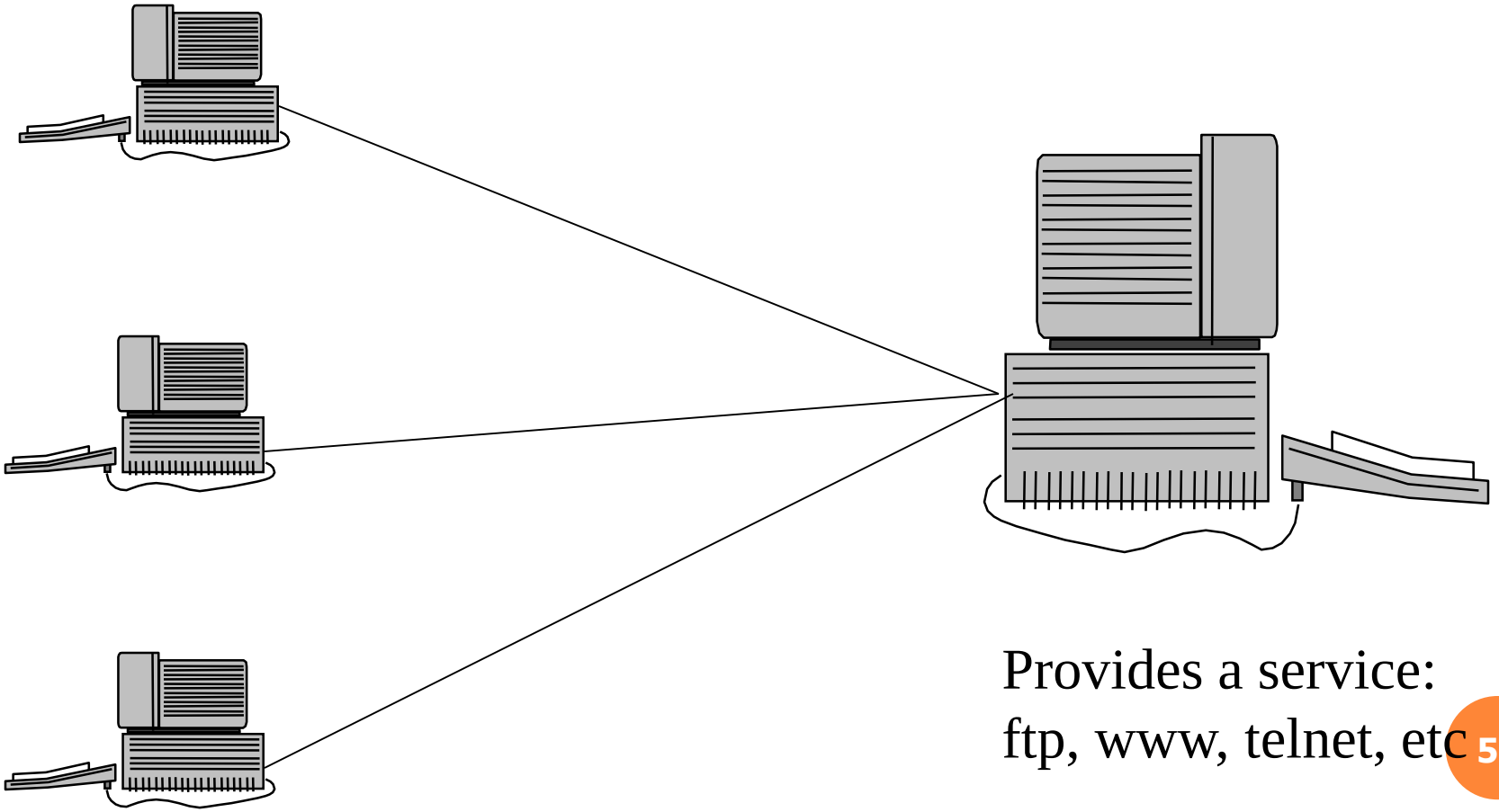
DatagramSocket - a UDP socket

InetAddress - an internet address

Networking with Java

Client/Server Model

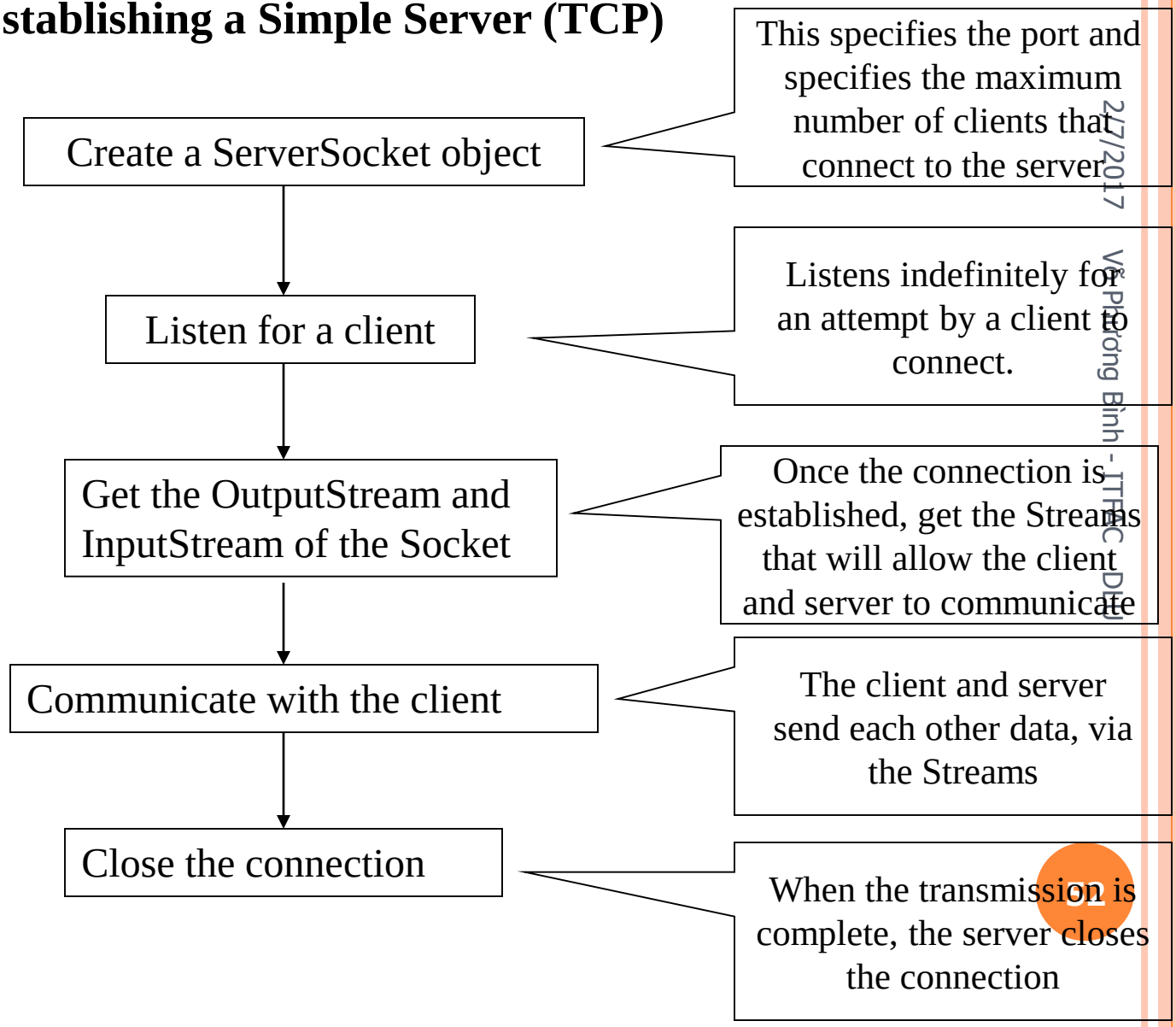
Clients



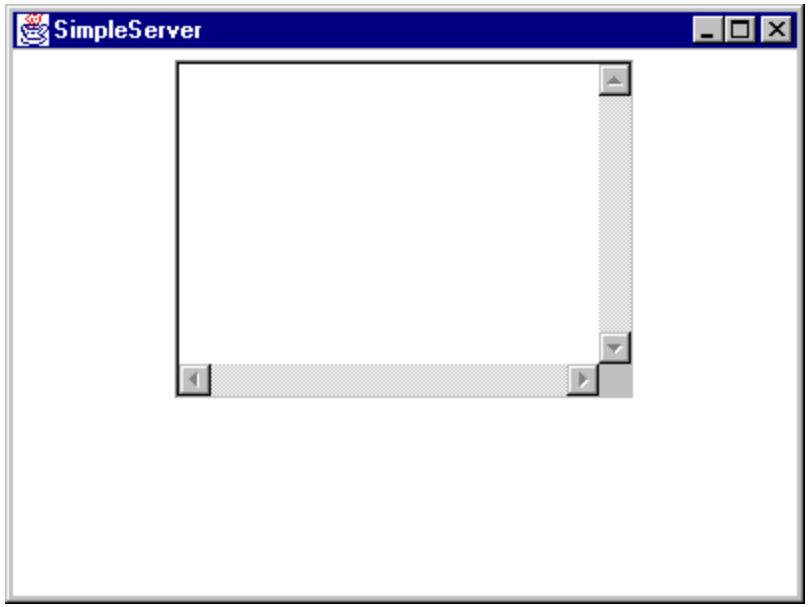
Provides a service:
ftp, www, telnet, etc

Networking with Java

Establishing a Simple Server (TCP)



Create a server that listens on port 8000. The client will send Strings. Display the Strings in the TextArea



```
import java.net.*;
import java.awt.*;
import javax.swing.*;
import java.io.*;
public class SimpleServer extends JFrame {
    JTextArea display;
    public SimpleServer( ) {
                super(“Simple Server”);
                Container con = getContentPane( );
                display = new JTextArea(10,30);
                display.setEditable(false);
                con.add(display);
                setSize(400,300);
                setVisible(true);
    }
    public void startServer( ) {
        ServerSocket server=null;
        Socket connection=null;
        String message;
       DataInputStream input;
        try {
                server = new ServerSocket (8000);
               while( true) {
               connection = server.accept ();
              display.append("Connection established\n");
             input = new DataInputStream (connection.getInputStream () );
               message = input.readUTF();
               display.append("Received message: "+message+"\n");
               connection.close();
            }
        } catch (IOException e) { e.printStackTrace(); }
     }
```

Create a server socket.the server will listen on port 8000

Listen for a client

Once the connection is established, get the Streams that will allow the client and server to communicate

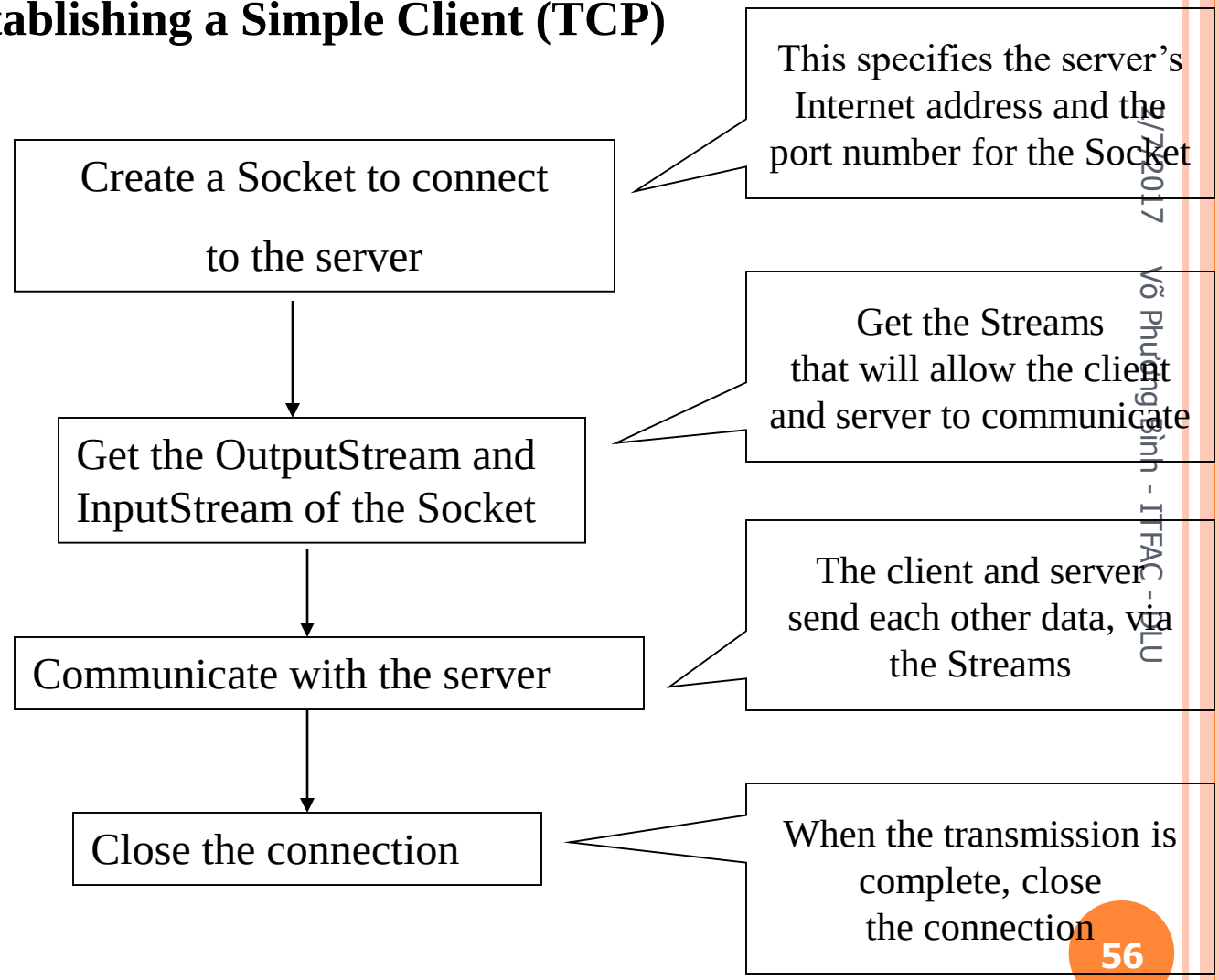
Communicate with the client

Close the connection

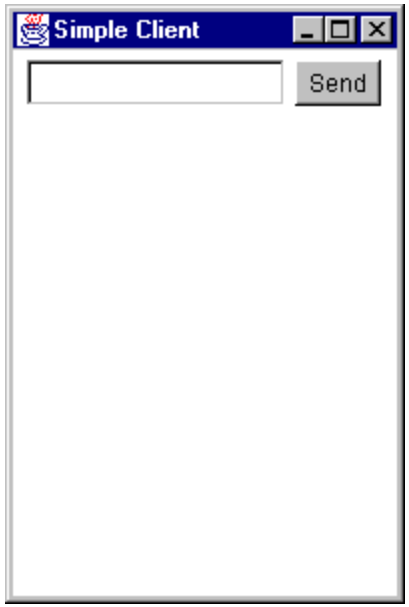
```
public static void main(String args[] ) {

        SimpleServer s = new SimpleServer( );
         s.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
         s.startServer();
    }

}
```

Networking with Java

Establishing a Simple Client (TCP)



Create a client that sends data to the Simple Server. The user types text in the TextField and then presses the Send Button. The text in the TextField is sent to the Server. The Server is listening on port 8000.



```
import java.awt.*;
import java.net.*;
import java.awt.event.*;
javax.swing.*
import java.io.*;

public class SimpleClient extends JFrame implements ActionListener{

    JTextField sendThis;
    JButton send;
    public SimpleClient( ) {

                super("Simple Client");
                Container con = getContentPane( );
                con.setLayout(new FlowLayout( ) );
                sendThis = new JTextField(15);
                con.add(sendThis);
                send = new JButton("Send");
                send.addActionListener(this);
                con.add(send);
                setSize(200,300);
                setVisible(true);
    }

    public void actionPerformed(ActionEvent action) {
        Socket client = null;
        DataOutputStream output;
        try {
            // client = new Socket( InetAddress.getByName("kneedeep.cis.famu.edu"),8000);
            client = new Socket( InetAddress.getLocalHost(),8000);
            output = new DataOutputStream( client.getOutputStream( ) );
            output.writeUTF( sendThis.getText( ) );

          } catch(IOException e) { e.printStackTrace(); }

    }
```

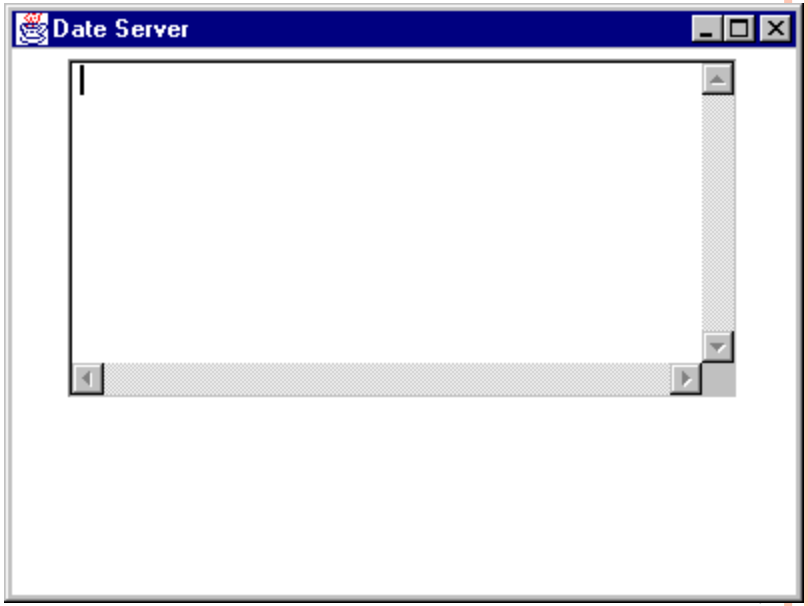
Create a Socket using the port the server is listening on

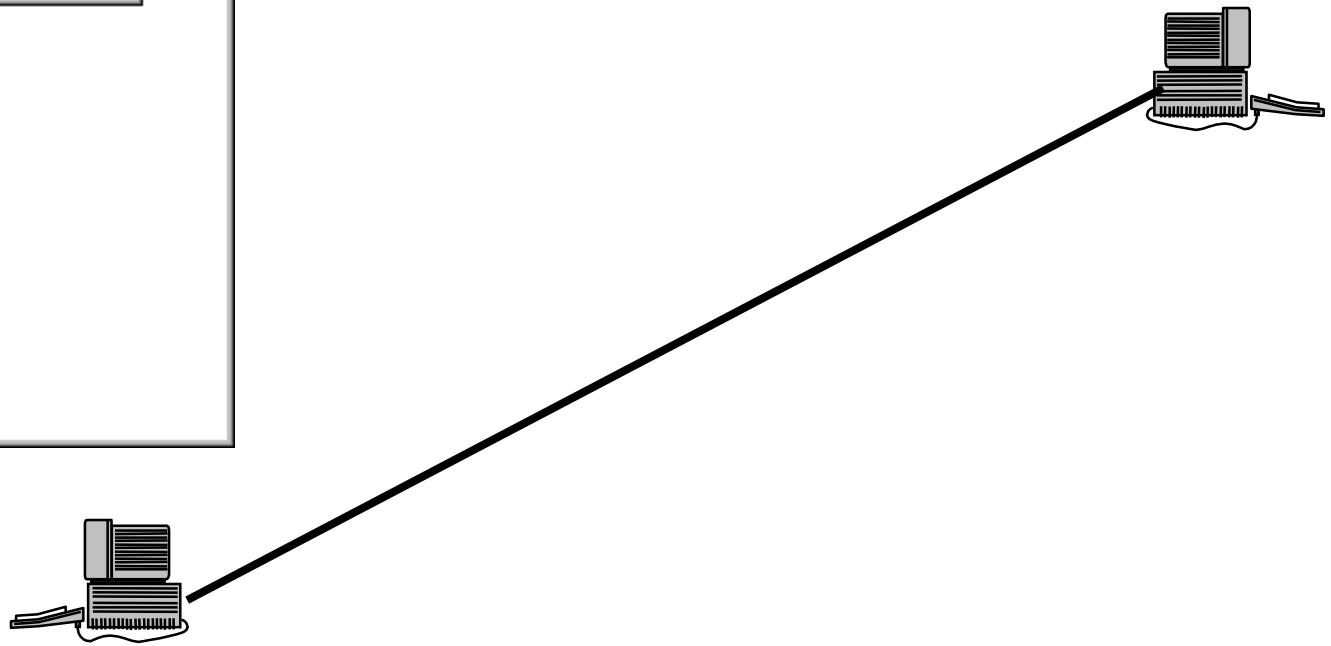
Get the Streams

Communicate with the client

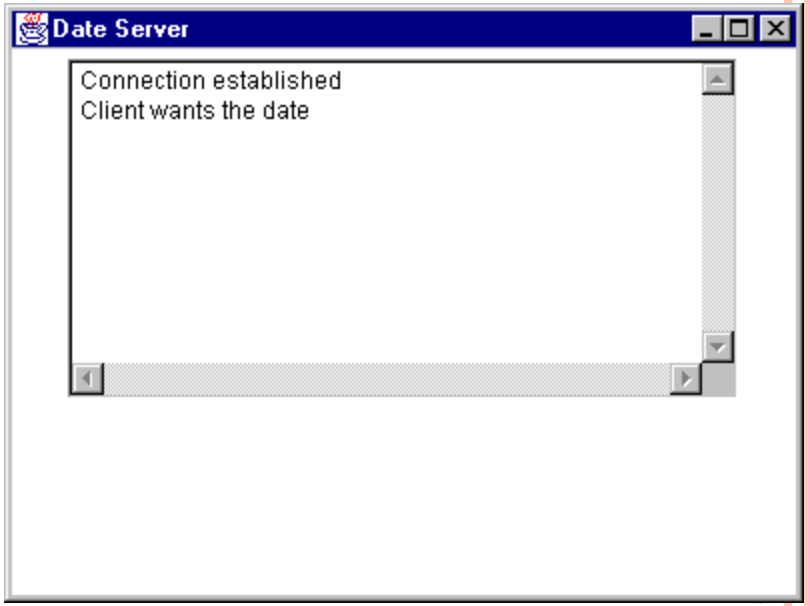
```
public static void main(String args[ ] ) {

        SimpleClient c = new SimpleClient();
       c.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
    }


}
```

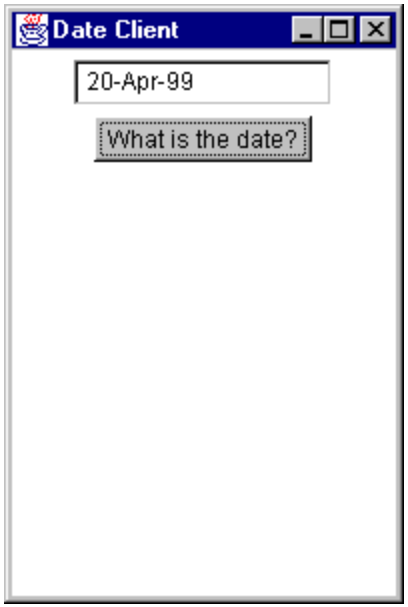
Create a server that returns the current Date to a client. Create the client that requests the current Date.

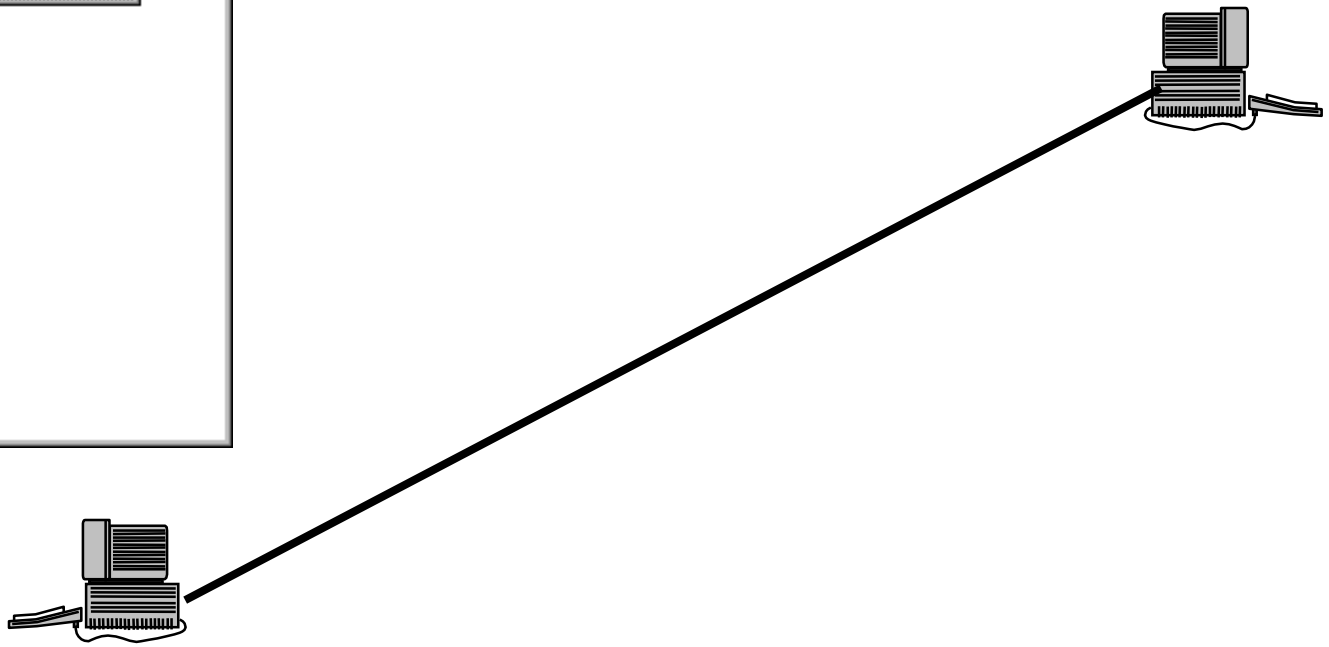




Create a server that returns the current Date to a client. Create the client that requests the current Date.







```
import java.awt.*;
import java.net.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

public class DateClient extends JFrame implements ActionListener{
    JTextField theDate;
    JButton getDate;
    public DateClient( ) {
        super("Date Client");
        Container con = getContentPane( );
        con.setLayout( new FlowLayout( ) );
        theDate = new JTextField(15);
        con.add(theDate);
        getDate = new JButton("What is the date?");
        getDate.addActionListener(this);
        con.add(getDate);
        setSize(200,300);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent action) {
      String date;
      Socket client = null;
      DataOutputStream output;
      DataInputStream input;
      try {

//       client = new Socket( InetAddress.getByName("kneedeep.cis.famu.edu"),8000);
        client = new Socket( InetAddress.getLocalHost(),8000);
        output = new DataOutputStream( client.getOutputStream() );
        output.writeUTF("Give me the date");
        input = new DataInputStream( client.getInputStream() );
        date = input.readUTF( );
        theDate.setText( date );
      } catch(IOException e) { e.printStackTrace(); }
   }
```

```
public static void main(String args[ ] ) {

        DateClient c = new DateClient();
      c.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
   }

}
```

The Server is next . . .

```
import java.net.*;
import java.awt.*;
import java.io.*;
import java.util.*;
import java.text.*;
import javax.swing.*;

public class DateServer extends JFrame {
    JTextArea display;

    public DateServer( ) {
                super("Date Server");
                Container con = getContentPane( );
                display = new JTextArea(10,45);
                display.setEditable(false);
                con.add(display);
                setSize(400,300);
                setVisible(true);

    }

    public void startServer( ) {

        ServerSocket server=null;
        Socket connection=null;
        String message;
                    DataInputStream input;
                    DataOutputStream output;
        try {
            server = new ServerSocket (8000);
            while( true) {

              connection = server.accept ();
              display.append("Connection established\n");
              input = new DataInputStream (connection.getInputStream () );
              output = new DataOutputStream (connection.getOutputStream ( ) );
```

```
            message = input.readUTF();
            if( message.equals("Give me the date") ) {
                  display.append("Client wants the date\n");
             String dateString = DateFormat.getDateInstance().format( new
Date() );
             output.writeUTF( dateString );
            }
            connection.close();
         }

       } catch (IOException e) { e.printStackTrace(); }


    }
    public static void main(String args[] ) {

            DateServer s = new DateServer( );
            s.startServer();
    }


}
```