Java Servlets

St. Louis Java Special Interest Group

March, 1999

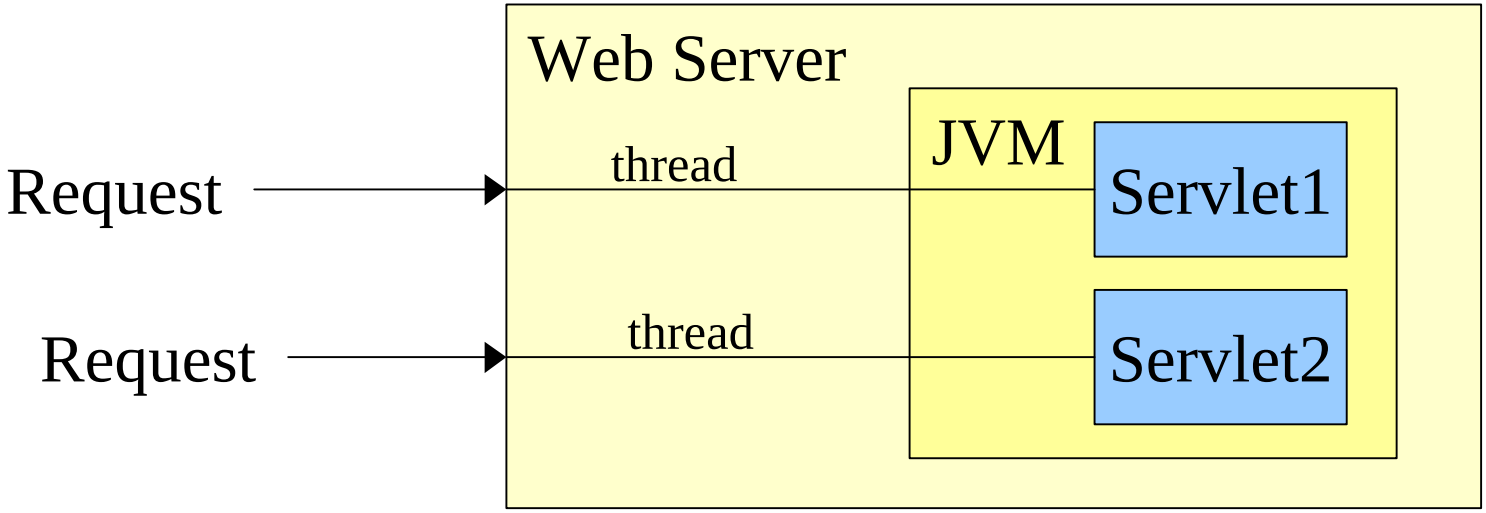
Eric M. Burke
Sr. Software Engineer
Object Computing, Inc.
http://home.fiastl.net/ericb/

What are Applets?

- Web browser “extensions”
  - Applets share a single Java Virtual Machine, embedded in the web browser
- Advantages of Applets
  - complex GUIs are possible
    - HTML and JavaScript are very limited
  - sophisticated processing can occur on the client
- Disadvantages of Applets
  - long initial download time
  - browser incompatibilities
    - AWT 1.1 event model was only introduced in 4.x browsers
  - bugs in various Java implementations
  - firewall restrictions

What are Servlets?

- A Servlet is a generic **server** extension
  - we will focus on **web server** extensions, although Servlets could be used to extend any sort of server
  - a Servlet-enabled server has a single Java Virtual Machine
    - each instance of a Servlet runs within this JVM
    - each client request is handled by a different thread
      - a thread requires far fewer resources than a process, as in CGI



What are Servlets? (Cont'd)

- The Servlet API is a **standard extension** to Java, produced by Sun
  - javax.servlet.*
  - javax.servlet.http.*
- Client browser does not have to support Java
  - Servlets are implemented entirely on the web server
  - all the client browser sees is HTML
- Disadvantages
  - Java Servlet API is relatively new, and changing rapidly
    - see upcoming slide on web server support

Servlet Advantages

- Performance
  - Servlets are faster than CGI because Servlets use threads rather than processes
- Portability
  - Servlets are just as portable as any Java application
- Reliability
  - Servlets *generally* will not crash the web server because they run inside of a Java Virtual Machine
- Simplicity
  - Servlets are very easy to implement
- Power
  - all standard Java APIs are available: JDBC, RMI, JavaMail, etc…

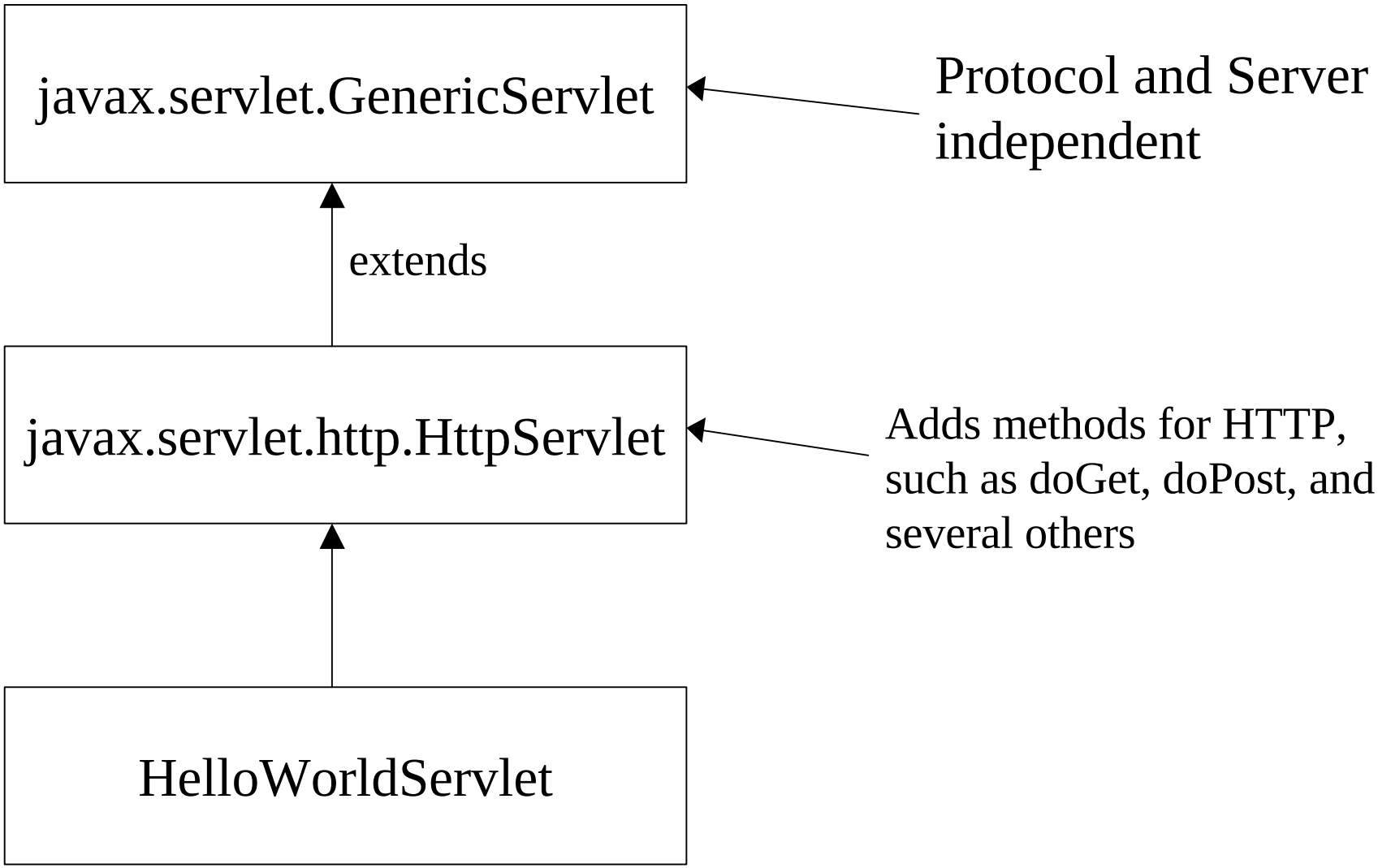
Web Server Support for Servlets

- Servlets will run on every major web server
  - either natively, or by using a 3rd party **servlet engine**
  - 3rd party servlet engines are useful if
    - your web server does not support Servlets natively
    - your web server is out of date
      - it may only support version 1.x of the Servlet API, instead of 2.x
- Popular servlet engines include:
  - JServ, a free engine for Apache: http://java.apache.org/
  - JRun, free and commercial versions available for all major web servers: http://www.livesoftware.com/
    - this engine seems to be the most widely used
    - the “free” version is limited to non-commercial use

Java Servlet Development Kit (JSDK)

- Free from Sun
- Includes:
  - source code for javax.servlet and javax.servlet.http
  - **servletrunner**, a Servlet testing utility
  - tutorial and API documentation
  - jsdk.jar for running Servlets with JDK 1.1.x
- The javax.servlet and javax.servlet.http packages are included with JDK 1.2

HelloWorldServlet Architecture



HelloWorldServlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
        String name = req.getParameter("name");
        if (name == null) { name = "World"; }

        resp.setContentType("text/html");
        PrintWriter out = new PrintWriter(resp.getOutputStream());
        out.println("<HTML>");
        out.println("<HEAD><TITLE>HelloWorld Output</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<H1>Hello, " + name + "!</H1>");
        out.println("</BODY>");
        out.println("</HTML>");
        out.close();
    }
}
```

HelloWorldServlet (Cont'd)

- To test HelloWorldServlet, run **servletrunner**

```
servletrunner -d C:\test\helloworld -s
  C:\test\helloworld\helloworld.properties
```

  - helloworld.properties is a property file with one line

```
servlet.helloworld.code=HelloWorldServlet
```

- Connect to the servlet using a web browser
  - http://localhost:8080/servlet/helloworld
    - "helloworld" is an alias to the HelloWorldServlet class
    - 8080 is the port number that servletrunner defaults to
  - http://localhost:8080/servlet/helloworld?name=Eric
    - passes "Eric" as the name
  - http://localhost:8080/servlet/HelloWorldServlet?name=Eric+Burke
    - passes "Eric Burke" as the name (+ is used for spaces)
    - the class name was used instead of the alias; either works

The Servlet Sandbox

- Servlets are either trusted or untrusted
  - a trusted Servlet has unrestricted access to web server resources
  - an untrusted Servlet is limited by a SecurityManager object
    - java.lang.SecurityManager limits access to many actions
      - reading and writing files, establishing network connections, starting processes, etc.
      - this is the same approach that Applets use
      - different web servers will implement this differently, allowing different degrees of control
    - locally installed Servlets typically have fewer restrictions
    - a “remote” Servlet is one that is not installed on the web server machine
      - a more restrictive SecurityManager is typically used
      - digital certificates can be used to create signed Servlets, just like signed Applets

javax.servlet.Servlet Interface

- Represents a Java class that runs within a network service, usually a web server
  - implementing classes are javax.servlet.GenericServlet and javax.servlet.http.HttpServlet
  - Clients request services from the Servlet
- Important methods
  - init(), destroy()
    - perform initialization and cleanup
  - getServletInfo()
    - returns a one line description of this Servlet
  - getServletConfig()
    - returns a ServletConfig object
  - service()
    - handles incoming requests from clients

javax.servlet.GenericServlet Class

- Implements Servlet and ServletConfig
- Servlets will typically extend this class or HttpServlet
  - if a Servlet already extends some other class, it can implement the Servlet interface instead
- Important methods
  - init(), destroy()
    - perform resource allocation and deallocation
    - you may want to override these to establish a database connection, for example
  - getServletInfo
    - return a single line String description of this Servlet
  - service(ServletRequest, Servlet Response)
    - the only abstract method - **see next slide...**

GenericServlet.service()

- Carries out a single request from a client

```
public abstract void service(ServletRequest req,
    SerletResponse res) throws ServletException, IOException;
```

- ServletRequest contains parameters from the client
- Output is sent to the ServletResponse
- Servers will not call this method until init() finishes
- Subclasses must be thread-safe
  - access to shared resources must be synchronized
  - one “brute force” solution is to synchronize this entire method
  - another solution is to implement the javax.servlet.SingleThreadModel interface
    - this guarantees that no two threads will concurrently invoke service() -- different servers will implement this differently

javax.servlet.ServletRequest Interface

- Provides information from the client to the server
  - passed as an argument to the Servlet.service() method
- Important methods
  - getReader()
    - returns a reference to a BufferedReader, for transferring text data
  - getInputStream()
    - allows the client to transfer binary data to the Servlet
  - lots of other methods
    - getAttribute, getCharacterEncoding, getContentLength, getParameter, getParameterNames, getParameterValues, getProtocol, getRealPath, getRemoteAddr, getRemoteHost, getScheme, getServerName, getServerPort
- HttpServletRequest extends this interface
  - adds several additional HTTP-specific methods

javax.servlet.ServletResponse Interface

- Allows the Servlet to return data to the client
  - MIME data types are used
  - see RFC 2045
- Important methods
  - getOutputStream() - for binary data
  - getWriter() - for text data
  - setContentType(String type)
    - usually, type will be “text/plain” or “text/html”
    - this method **MUST** be called **BEFORE** calling getWriter() or before using the output stream returned by getOutputStream()
  - setContentLength()
    - this can improve efficiency if you know how many bytes you will be sending to the client

javax.servlet.http.HttpServlet Class

- An abstract class that supports HTTP Servlets
- Important methods
  - HTTP GET, POST, PUT, DELETE
    - doGet
    - doPost
    - doPut
    - doDelete
  - getServletInfo
    - subclasses should return a one line String description
  - init, destroy
    - subclasses may optionally override
    - useful to initialize and destroy database connections or other “expensive” resources

The HttpServlet.service() Method

- An abstract method in GenericServlet, implemented by HttpServlet

```
void service(ServletRequest req, ServletResponse res)
                    throws ServletException, IOException
```

  - web server directs incoming requests to the service() method
  - the service() method figures out the type of request and forwards to a specific method
    - for example, HTTP “GET” requests are forwarded to the doGet() method
  - this method is rarely overridden
    - instead, you will override, doGet(), doPost(), etc...

javax.servlet.http.HttpServletRequest

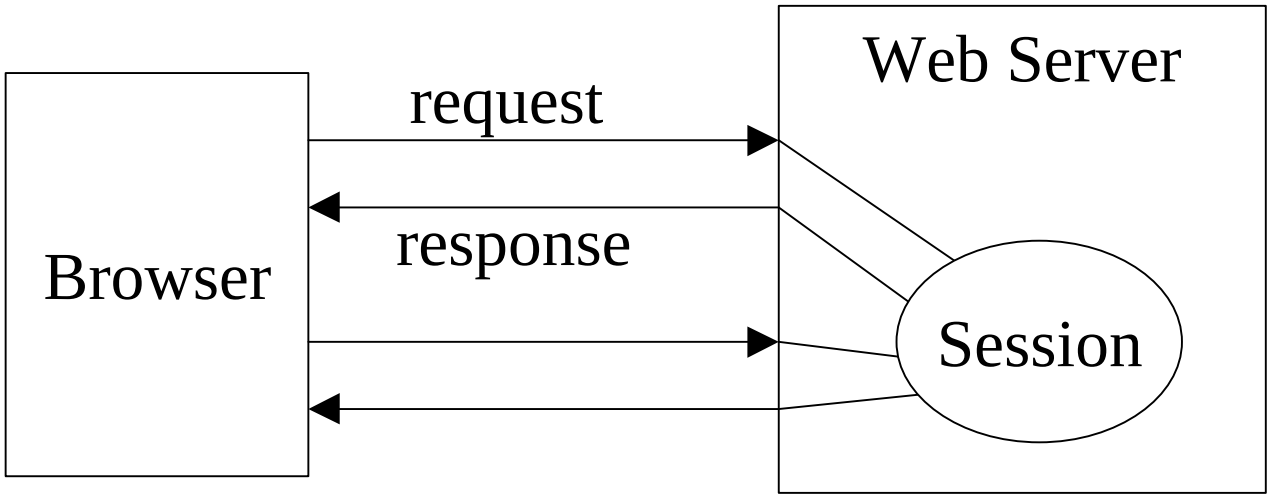
- Extends javax.servlet.ServletRequest
  - getInputStream, getReader, getParameterNames, etc...
- Important methods
  - getCookies
    - returns an array of Cookie objects
  - getMethod
    - same as CGI REQUEST_METHOD, may return GET, POST, PUT
  - getSession
    - return an HttpSession object, or create a new one
  - several other methods
    - see JavaDocs
    - several methods for determining the Servlet name, the URL path, and the query string

javax.servlet.http.HttpServletResponse

- Extends javax.servlet.ServletResponse
  - getOutputStream, getWriter, setContentType
- Several useful HTTP constants are defined in this interface
  - see JavaDocs
  - example: SC_FORBIDDEN (403)
- Important methods
  - sendError(int statusCode)
    - see following page
  - sendRedirect
    - send an alternate URL to the client
  - addCookie
  - several other methods
    - see JavaDocs

Session Tracking

- Web servers need a way of tracking users as they navigate from screen-to-screen in a web-based app
  - this is known as Session Tracking
  - this would allow “screen 2” to remember the username and password that was entered on “screen 1”
- The concept of Session Tracking can be implemented in many different ways



(Logical Model)

javax.servlet.http.HttpSession Interface

- Simplifies session management
  - works with either Cookies or URL rewriting
- Allows arbitrary Java objects to be associated with client sessions
- Important methods
  - putValue, removeValue, getValue, getValueNames
    - allow objects to be associated with Strings, similar to Hashtable
  - getCreationTime, getLastAccessedTime
    - returns the age of a session
  - invalidate
    - allows you to destroy a session

Session Example

```
protected void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
    // “true” causes a new session to be created if needed
    HttpSession session = req.getSession(true);

    // require the user to logon before they can access this Servlet.
    // In this example, the String “authentication” is put into the
    // Session by a Logon page
    if (session.getValue(“authentication”) == null) {
        res.sendRedirect(logonURL);
    } else {
        // normal processing here
    }
}
```

Applet-to-Servlet Communication

- HTTP is the best approach for public Internet use
- Advantages
  - works behind firewalls
  - relatively easy to implement
  - can work with pre-JDK 1.1 browsers
    - the examples in this presentation require JDK 1.1 because of Serialization
- Disadvantages
  - as in other approaches, only the Applet can initiate a conversation with the Servlet
    - you can also use Sockets or RMI
  - performance is not as fast as with raw Sockets
    - multiple requests require multiple subsequent connections to the server

A Framework for HTTP Communication

- com.ociweb.applet2servlet package
  - ObjectServerI
    - an **interface** which describes the server side of the connection
    - allows one Serializable object to be sent and another returned
  - ObjectServer
    - a class that implements ObjectServerI
    - Servlets instantiate an ObjectServer and use it to communicate with an Applet
  - ObjectClientI
    - an **interface** which describes the client side of the connection
  - ObjectClient
    - a class that implements ObjectClientI
    - Applets instantiate an ObjectClient instance and use it to communicate with a Servlet

ObjectServerI Interface

```
package com.ociweb.applet2servlet;
import java.io.*;

/**
 * A Servlet may use this interface to communicate with an Applet.
 * This protocol allows one object to be sent and one object to
 * be returned per connection.
 * @author Eric M. Burke, Object Computing, Inc.
 */
public interface ObjectServerI {
    /**
     * Read a request from the Applet.
     * @return an object passed from the Applet.
     */
    Serializable read() throws IOException, ClassNotFoundException;

    /**
     * Write a response to the Applet.
     * @param obj the data to send back to the Applet.
     */
    void write(Serializable obj) throws IOException;
}
```

ObjectServer Class

```
package com.ociweb.applet2servlet;

import java.io.*;

/**
 * A concrete implementation of the ObjectServerI interface.
 * @author Eric M. Burke, Object Computing, Inc.
 */
public class ObjectServer implements ObjectServerI {
    private InputStream in;
    private OutputStream out;


    /**
     * @param in the stream that will provide one object from the client.
     * @param out the stream that will allow this class to return an
     * object to the client.
     */
    public ObjectServer(InputStream in, OutputStream out) {
        this.in = in;
        this.out = out;
    }
```

ObjectServer Class (Cont'd)

```
    // Read a request from the Applet.
    public Serializable read() throws IOException, ClassNotFoundException {
        ObjectInputStream ois = new ObjectInputStream(in);
        try {
            // readObject() may throw ClassNotFoundException
            return (Serializable) ois.readObject();
        } finally {
            ois.close();
        }
    }

    // Write a response to the Applet.
    public void write(Serializable obj) throws IOException {
        ObjectOutputStream oos = new ObjectOutputStream(out);
        try {
            oos.writeObject(obj);
        } finally {
            oos.close();
        }
    }
}
```

ObjectClientI Interface

```
package com.ociweb.applet2servlet;

import java.io.IOException;
import java.io.Serializable;

/**
 * An Applet may use this interface to communicate with a Servlet.
 * This protocol allows one object to be sent and one object to
 * be returned per connection.
 * @author Eric M. Burke, Object Computing, Inc.
 */
public interface ObjectClientI {
    /**
     * @param obj the object to send to the Servlet.
     * @return the result object from the Servlet.
     * @throws IOException if anything went wrong with the connection.
     */
    Serializable write(Serializable obj) throws IOException,
            ClassNotFoundException;
}
```

ObjectClient Class

```
package com.ociweb.applet2servlet;

import java.io.*;
import java.net.*;

/**
 * This is a concrete implementation of the ObjectClientI interface.
 * @author Eric M. Burke, Object Computing, Inc.
 */
public class ObjectClient implements ObjectClientI {
    private URL serverURL;

    public ObjectClient(String serverURL) throws MalformedURLException {
        this(new URL(serverURL));
    }

    public ObjectClient(URL serverURL) {
        this.serverURL = serverURL;
    }
```

ObjectClient Class (Cont'd)

```
/**
 * @param obj the object to send to the Servlet.
 * @return the result object from the Servlet.
 */
public Serializable write(Serializable obj) throws IOException,
ClassNotFoundException {
   URLConnection conn = serverURL.openConnection();
   conn.setDoOutput(true);   // allow writing to the connection
   conn.setUseCaches(false); // disable cache
   conn.setRequestProperty("Content-Type",
           "java-internal/" + obj.getClass().getName());

   ObjectOutputStream oos = new ObjectOutputStream(
           conn.getOutputStream());
   try {
       oos.writeObject(obj);
   } finally {
       if (oos != null) {
           oos.close();
       }
   }
```

ObjectClient Class (Cont'd)

```
        // return the response to the client
        ObjectInputStream ois = new ObjectInputStream(
                conn.getInputStream());
        try {
            // readObject() may throw a ClassNotFoundException
            return (Serializable) ois.readObject();
        } finally {
            if (ois != null) {
                ois.close();
            }
        }
    }
}
```

Example Applet

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

public class MyApplet extends java.applet.Applet implements ActionListener {
    private Button submitBtn = new Button("Submit");
    private TextField nameFld = new TextField(20);
    private TextField ageFld = new TextField(3);

    public void init() {
        setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        // … GUI Layout omitted
        submitBtn.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == submitBtn) {
            submit();
        }
    }
```

Example Applet (Cont'd)

```
    private void submit() {
        Person person = new Person(nameFld.getText(),
                                   Integer.parseInt(ageFld.getText()));
        try {

            ObjectClient client = new ObjectClient(
                    "http://localhost:8080/servlet/MyServlet");

            person = (Person) client.write(person);

            nameFld.setText(person.getName());
            ageFld.setText("" + person.getAge());
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Example Servlet

```
import com.ociweb.applet2servlet.ObjectServer;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * An example Servlet which shows one way to communicate with an Applet.
 * @author Eric M. Burke, Object Computing, Inc.
 */
public class MyServlet extends HttpServlet {
    /**
     * GET and POST will work equally well in this example.
     */
    public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
        doGet(req, res);
    }
```

Example Servlet (Cont'd)

```
    /**
     * This method expects the req object to provide a Person object.
     * The object will be modified, then returned to the client.
     */
    public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
        // construct the object which allows the communication
        ObjectServer srv = new ObjectServer(req.getInputStream(),
                                            res.getOutputStream());
        Serializable ser = null;
        try {
            ser = srv.read();  // get request
        } catch (ClassNotFoundException cnfe) {
            getServletContext().log(cnfe,
                                    "Could not read client request.");
        }
        if (ser instanceof Person) {
            Person person = (Person) ser;
            person.setName(person.getName().toUpperCase());  // modify some values
            person.setAge(person.getAge() + 1);
            srv.write(person);  // send response
        }
    }
}
```

JDBC

- Java Database Connectivity (JDBC)
  - allows Java programs to interact with relational databases
  - a thin API that requires knowledge of SQL
    - included with JDK 1.1 and above
    - java.sql package
  - an abstraction which makes database access portable
    - the device drivers are the only vendor-specific portion

Using JDBC with Servlets

- Create a subclass of HttpServlet
  - override the init() method
    - load the JDBC device driver
      - allows you to connect to a vendor's database
      - if this fails, throw an UnavailableException
        - » indicates to the web server that the Servlet cannot be started
        - » the Servlet should also write an entry to the log file
    - optionally create a Connection to the database
      - this could be done in the init method, or later in doGet or doPost
  - override doGet(), doPost(), or another method
    - create a JDBC Statement() object, which allows you to pass SQL to the database
    - format HTML results and return to the client
    - always use try/catch/finally to clean up after errors
      - a failed Servlet should not leave an open database connection

Further Topics…Not Enough Time!

- Server-Side Includes
- JavaServer Pages
- Cookies
- Generating GIFs using Servlets
- Processing HTML Forms
- Security
- Sending Email from Servlets

Recommended Reading

- Java Servlet Programming, O'Reilly
  - Jason Hunter with William Crawford
- HTML - The Definitive Guide, O'Reilly
  - Chuck Musciano & Bill Kennedy
- Servlet Interest mailing list
  - up to 100 messages on a busy day!
  - send email to: listserv@java.sun.com
    - the message body should contain:

```
SUBSCRIBE SERVLET-INTEREST [your real name]
```