Server Side Development: Servlets

ITM 602

Enterprise Application Development

Server Side Development

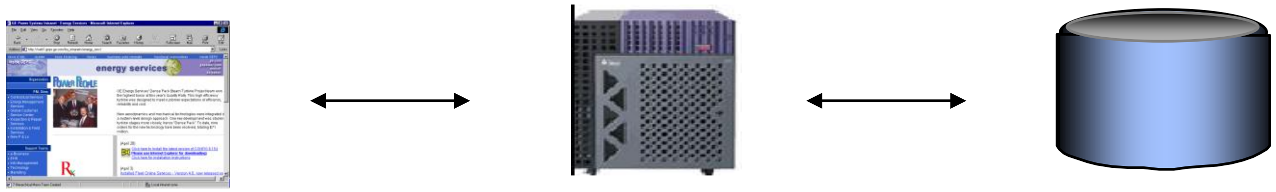
Outline

- Container Architecture
- Web Components
- Servlets and Servlet Applications
- Servlet API
  - Javax.servlet
  - Javax.servlet.http
- Deploying an Application

Server Side Development

Tiered Architecture

- The owl two-tiered client-erver model has been superceded by the multi-tiered architecture prevelant in the enterprise applications
  - Allows each layer to communicate just with layers above and below it
- Benefits of having a tiered application
  - Encapsulates rules and functionality together providing for easier maintenance & development
  - Enhances flexibility and reusability of logic and software components
  - Allows developers to focus on the area of their speciality e.g. database, servers, web page, etc.



Server Side Development

Web Server

- A web server is a program running on the server that listens for incoming requests and services those requests as they come in.
- Once the web server receives a request, depending on the type of request the web server might look for a web page, or it might execute a program on the server.
- It will always return some kind of results to the web browser, even if its simply an error message saying that it couldn't process the request.
- By default the role of a web server is to serve static pages using the http protocol
- Web servers can be made dynamic by adding additional processing capability to the server

Server Side Development

Server Extensions

- Several different tools are available for extending the server capabilities
  - Java enterprise architecture
  - VB .Net architecture
  - Active Server Pages (ASP)
  - CGI-Perl scripting
- These tools process incoming requests from the user and generate custom html pages

Server Side Development

Tomcat

- Tomcat is a stand alone web server and a servlet container
  - It is open source and free for usage
- It is written in Java
  - You do not have to be a Java programmer to use it
  - It's web server is not as fully featured as others like Apache
- Installing Tomcat
  - Make sure that jdk1.4 (or higher) is installed on your machine
  - Download the latest windows version of Tomcat
  - Run the installer by double clicking on the download
  - The installer checks if JRE and JDK are available for Tomcat
  - Accept the license agreement
  - Installation directory: c:\Program Files\Apache Tomcat 4.0
  - On installation you get a message *Completed*

HTTP

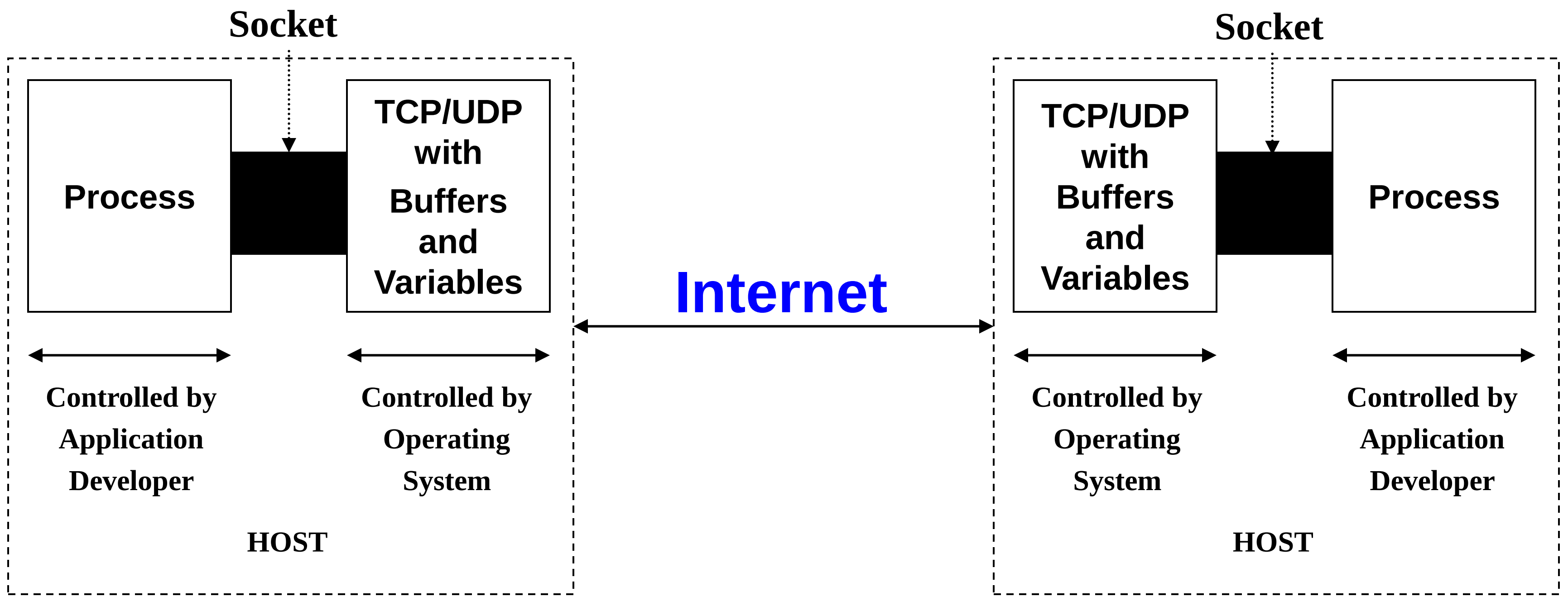
HTTP

Application Layer Protocol

- User applications implement this protocol
  - Other protocols implemented by the OS.
- Different applications use different protocols
  - Web Servers/Browsers use HTTP
  - File Transfer Utilities use FTP
  - Electronic Mail applications use SMTP
  - Naming Servers use DNS
- Interacts with transport layer to send messages

HTTP

Application Layer Protocol, cont'd.


Socket
Socket
Process
TCP/UDP with Buffers and Variables
Internet
TCP/UDP with Buffers and Variables
Process
Controlled by Application Developer
Controlled by Operating System
Controlled by Operating System
Controlled by Application Developer
HOST
HOST

- Two parameter required for identifying the receiving process
  - Host machine identifier - IP Address (localhost or ip-address)
  - Host machine process identifier - Port (80 or 8080 for web server)

HTTP
HyperText Transfer Protocol

- Lightweight protocol for the web involving a single request & response for communication
- Provides 8 methods
  - Get: Used to request data from server
    (By convention get will not change data on server)
  - Post: Used to post data to the server
  - Head: returns just the HTTP headers for a resource.
  - Put: allows you to "put" (upload) a resource (file) on to a webserver so that it be found under a specified URI.
  - Delete: allows you to delete a resource (file).
  - Connect:
  - Options: To determine the type of requests server will handle
  - Trace: Debugging

HTTP

GET and POST

- GET and POST allow information to be sent back to the web server from a browser
  - e.g. when you click on the "submit" button of a form the data in the form is send back to the server, as "name=value" pairs.
- Choosing GET as the "method" will append all of the data to the URL and it will show up in the URL bar of your browser.
  - The amount of information you can send back using a GET is restricted as URLs can only be 1024 characters.
- A POST sends the information through a socket back to the webserver and it won't show up in the URL bar.
  - This allows a lot more information to be sent to the server
  - The data sent back is not restricted to textual data and it is possible to send files and binary data such as serialized Java objects.

HTTP

HTTP Headers

- Contains information about client and the request
- Four categories of header information
  - General Information: Date, caching information, warnings etc.
  - Entity Information: Body of the request or response e.g. MIME type, length etc.
  - Request Information: Information about client e.g. cookies, types of acceptable responses etc.
  - Response Information: Information about server e.g. cookies, authentication information etc.
- General & Entity information used for both client & server
- Request information included by client
- Response information included by server

HTTP

Protocol

- HTTP is a stateless protocol
  - Request/Response occurs across a single network connection
  - At the end of the exchange the connection is closed
  - This is required to make the server more scalable
- Web Sites maintain persistent authentication so user does not have to authenticate repeatedly
- While using HTTP persistent authentication is maintained using a token exchange mechanism
- HTTP 1.1 has a special feature (keep-alive) which allows clients to use same connection over multiple requests
  - Not many servers support this
  - Requests have to be in quick succession

HTTP
Tracking State

- Three types of tracking methods are used:
  - Cookies: Line of text with ID on the users cookie file
  - URL Session Tracking: An id is appended to all the links in the website web pages.
  - Hidden Form Elements: An ID is hidden in form elements which are not visible to user
- Custom html page allows the state to be tracked

HTTP

HTTP Status Codes

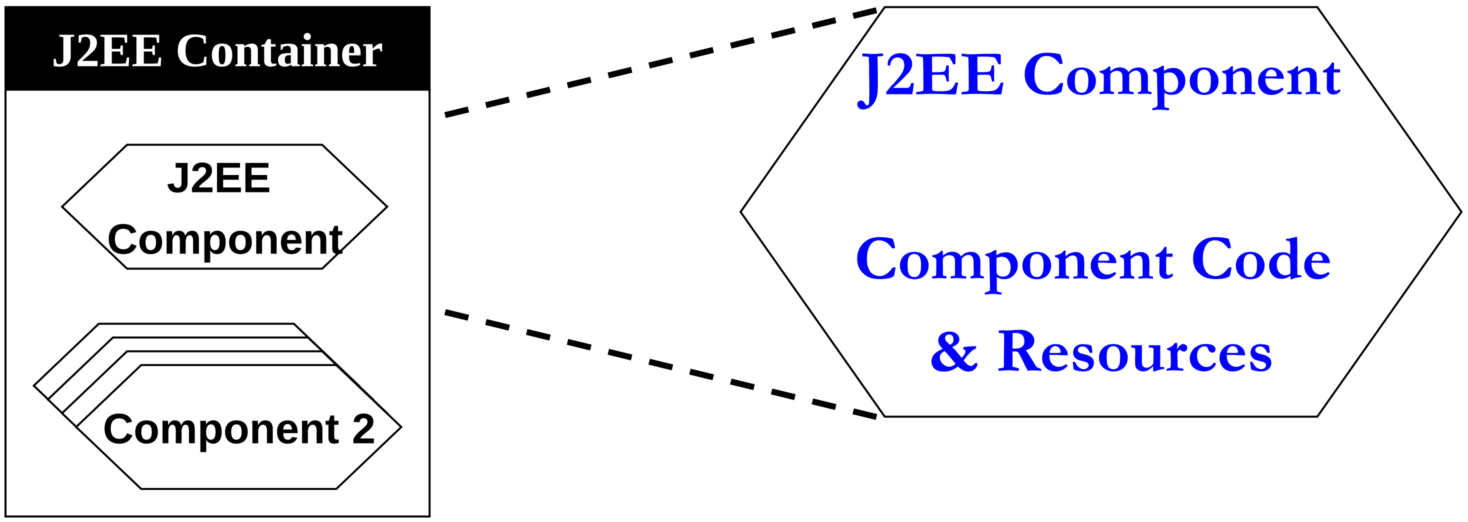
- When a server responds to a request it provides a status code
- Web Container automatically handles setting of status codes
- Five categories of status codes
  - Informational
  - Success
  - Redirection
  - Client error
  - Server error
- Common Status Codes
  - 200 – Request was processed normally
  - 401 – Unauthorized access
  - 403 – Forbidden
  - 404 – Requested resource not found on server
  - 405 – Method Not allowed
  - 500 – Internal server error

J2EE Architecture

J2EE Architecture

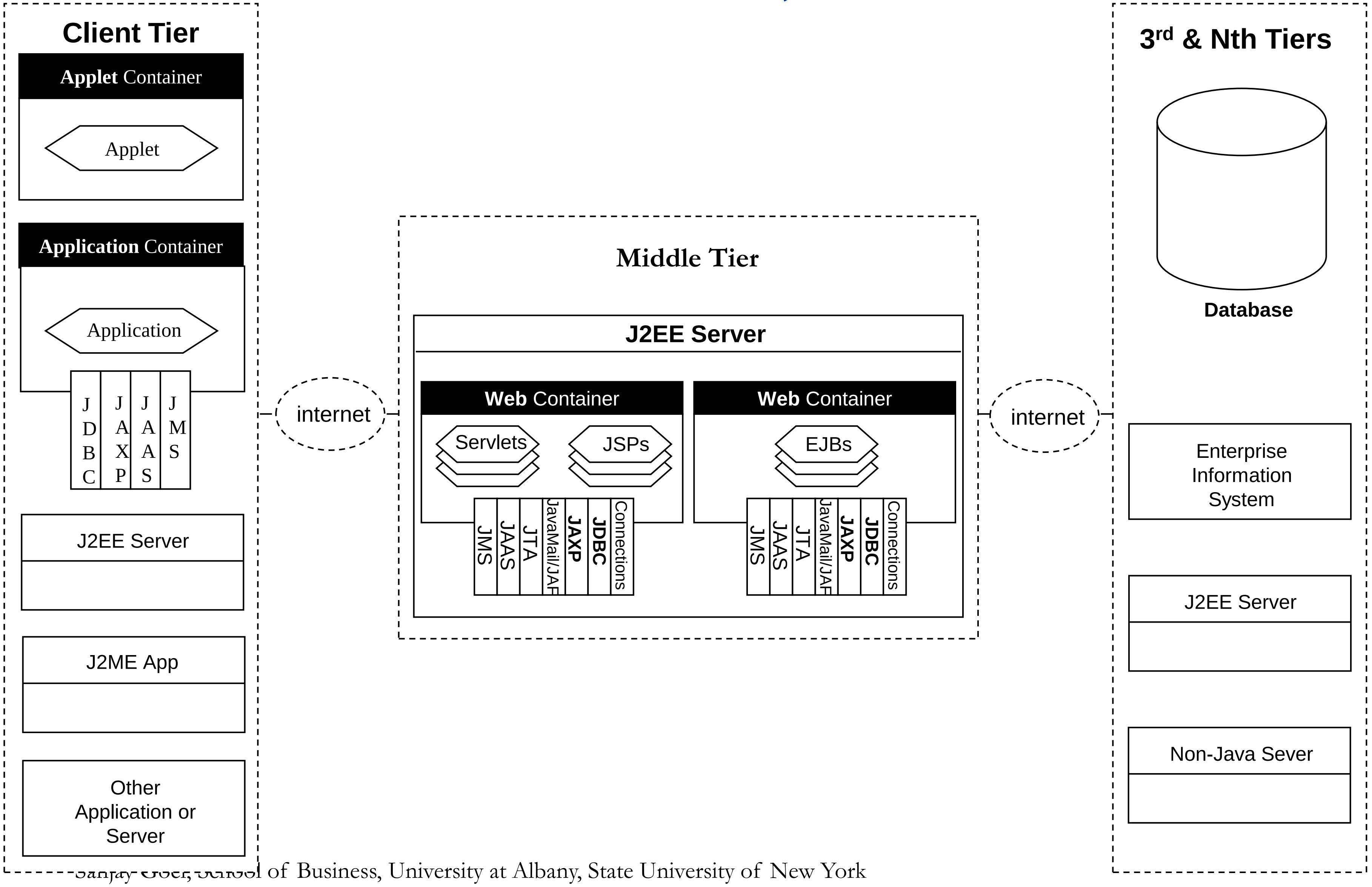
J2EE – Container Architecture

- Application is considered as a collection of related yet independent components
- Container acts as an execution environment for the components
- Container Provides services to the components



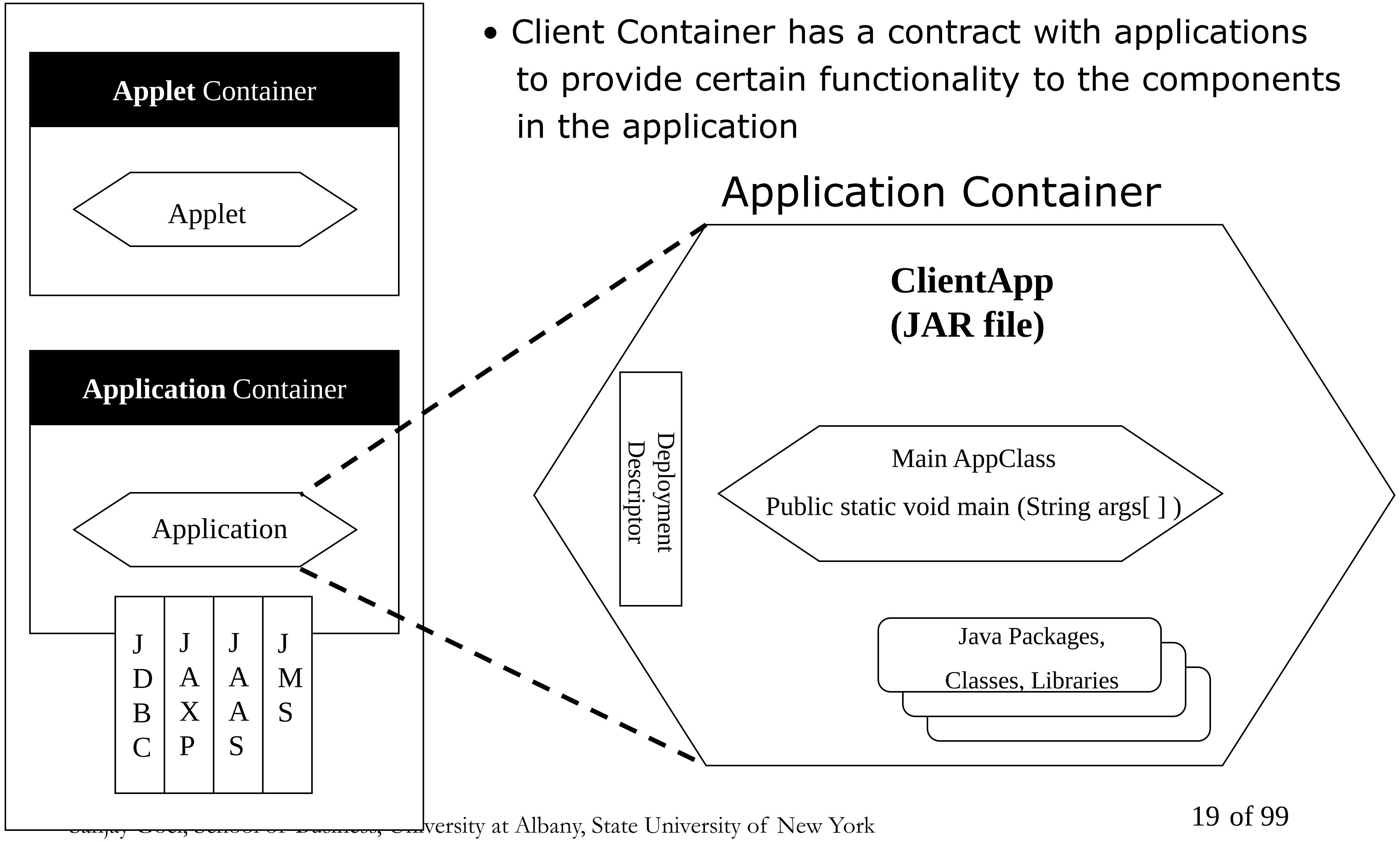
J2EE Architecture

J2EE – Container Architecture, cont'd.



J2EE Architecture
Client Tier

- Client Container has a contract with applications to provide certain functionality to the components in the application

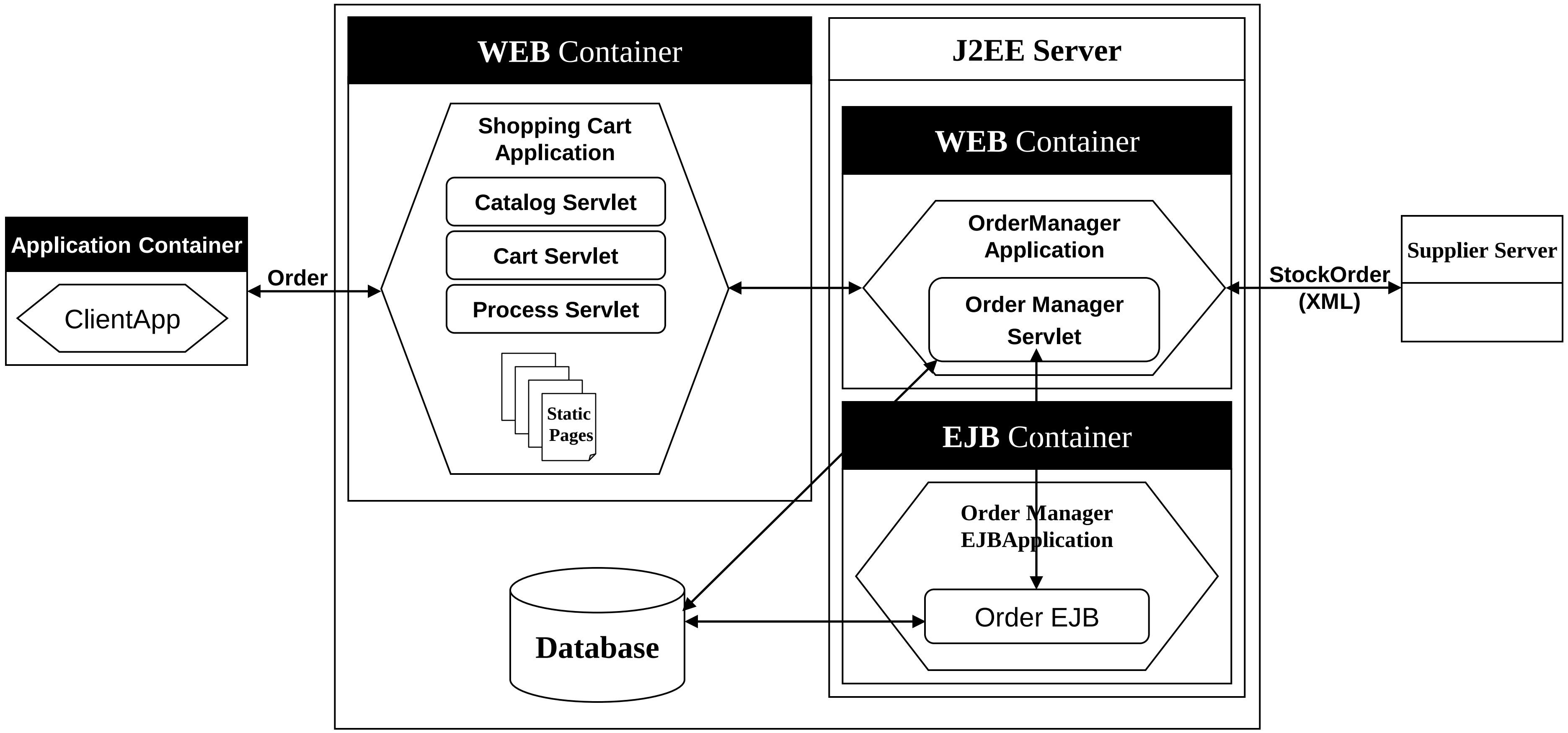


J2EE Architecture

Middle Tier Container

- Web Container
  - Manages execution of servlets and JSPs
  - Part of web or application server
  - Supports HTTP
- EJB Container
  - Business Components that contain business logic or rules
  - Two types of EJBs
    - Session Beans – Logic Oriented and deal with handling client requests and data processing
    - Entity Beand – Strongly coupled with data and deal with data access and persistence

J2EE Architecture
E-Commerce Scenario



- Two distinct parts of the applications
  - Shopping Cart: Handles consumer side of the store
  - Order Manager: Handles back end processing

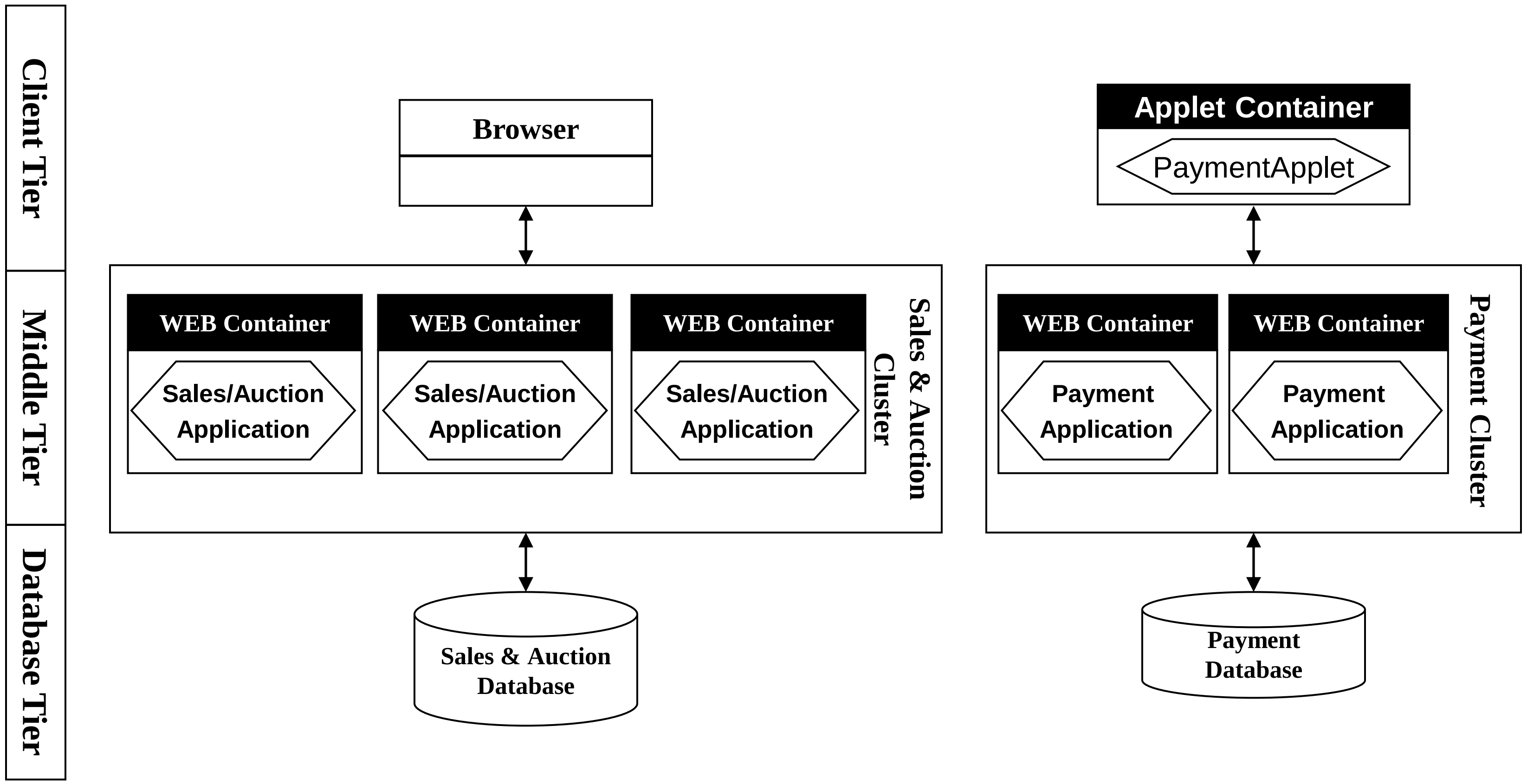
J2EE Architecture

E-Commerce

- Cart Application
  - Catalog servlet gets product data from the database
  - Cart servlet keeps track of the customer purchase
  - Process servlet processes the order
- Order Process Application
  - Processes customer order
  - Checks inventory levels (orders new parts from Suppliers)
  - Processes payments
  - Sends acknowledgement to the client

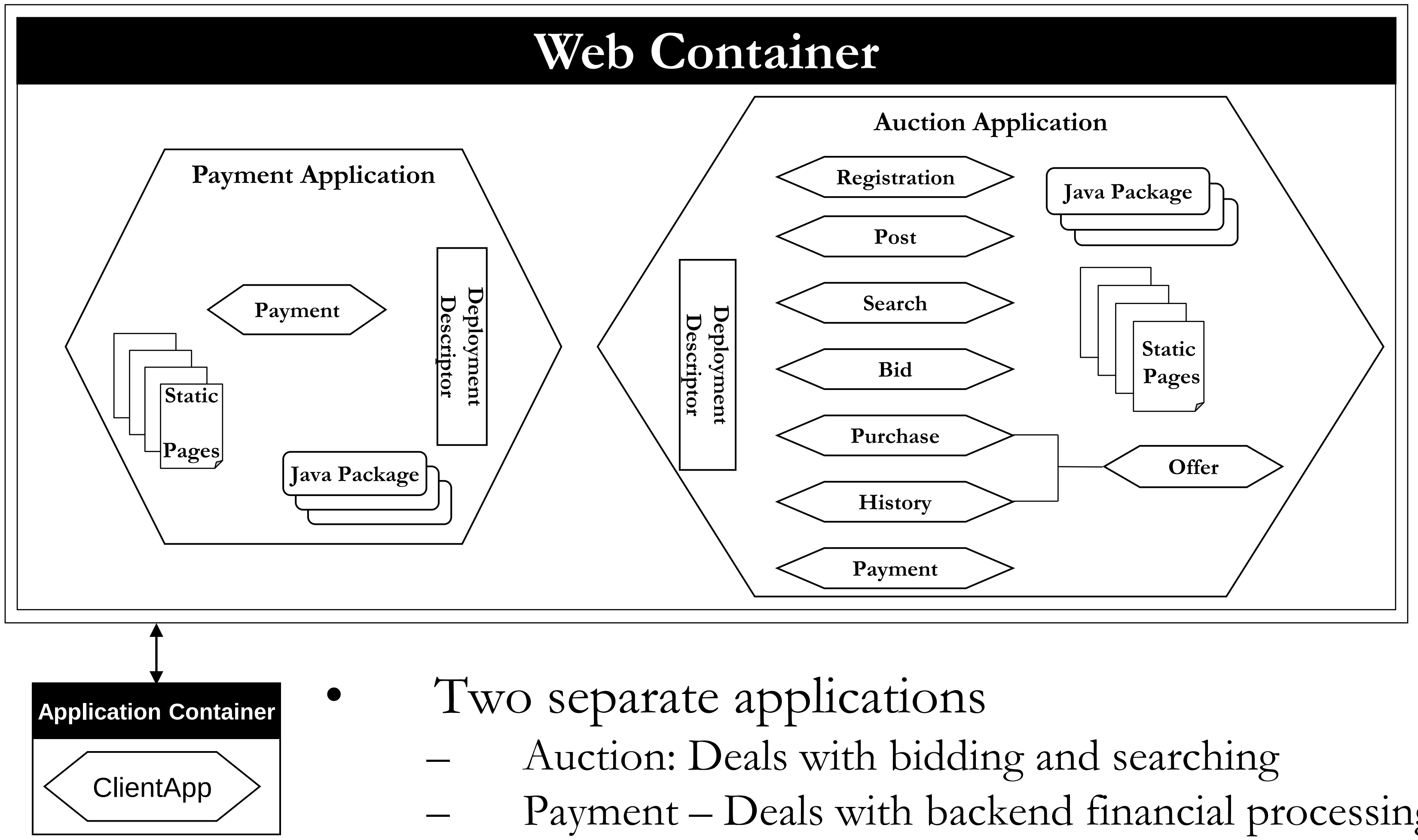
J2EE Architecture

E-Auctions



J2EE Architecture

E-Auctions – Container Ideas



- Two separate applications
  - Auction: Deals with bidding and searching
  - Payment – Deals with backend financial processing

J2EE Architecture
E-Auctions

- Client Side uses a web browser to view store
- Auction Application
  - Registration Servlet: Registers new users
  - Post servlet: Accepts new items for auction
  - Search servlet: Allows buyers to search database
  - Bid servlet: Allows users to bid on pending items
    - Informs the seller of the bid (e-mail)
  - Purchase servlet: Processes sales
  - History Servlet: Allows bidder/seller to review history of any item on auction
- Payment Application
  - Payment Servlet: Credits the buyer and Debits the seller (Credit card transactions)

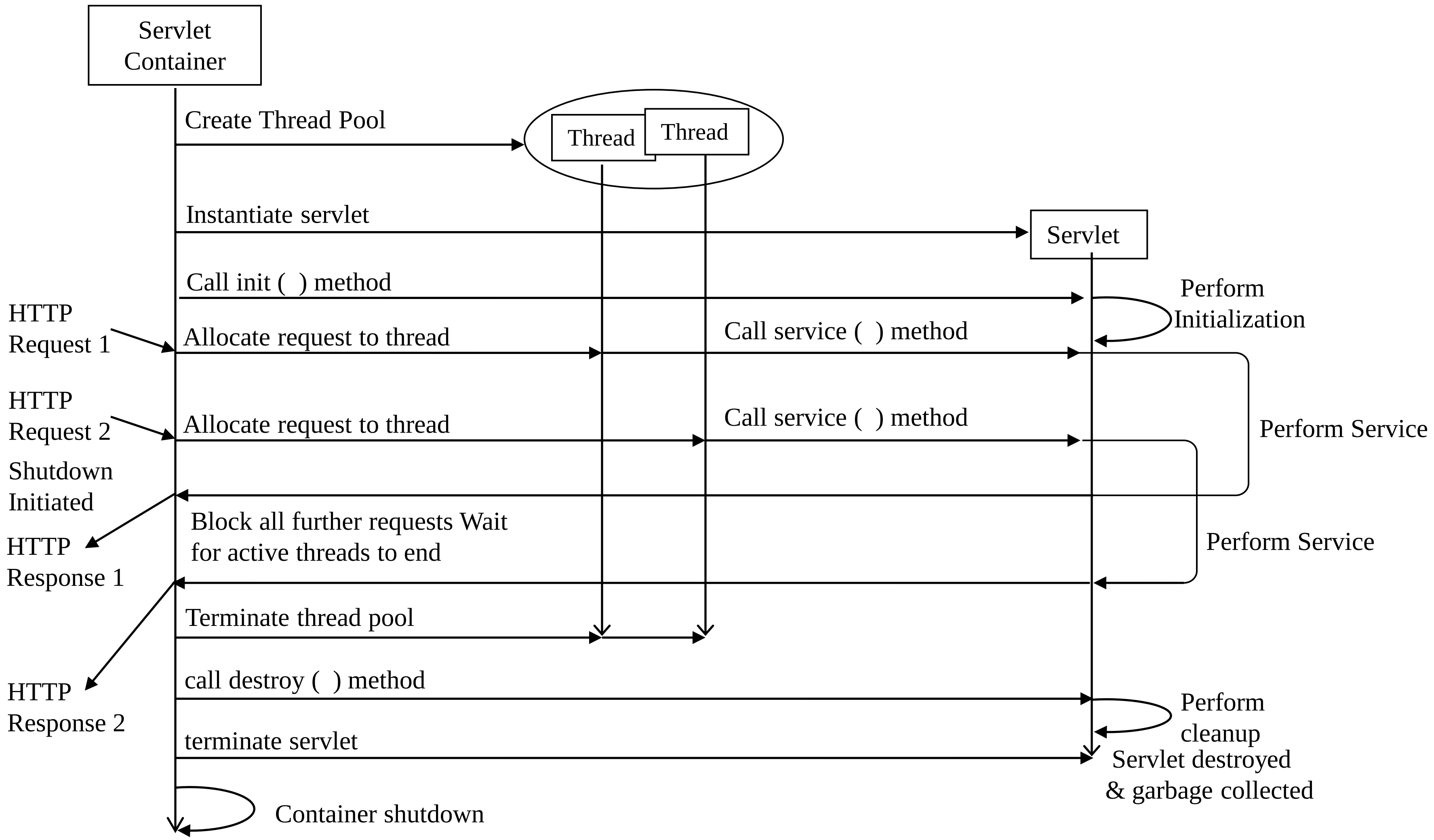
Servlets

Servlets
Introduction

- Classes that dynamically process requests and construct responses
  - Dynamically generate html pages in response to requests
  - May also send data in other forms like XML or serialized Java objects
  - Run in a servlet container and have access to services that the container provides
- In an application processing of each request will normally be done by a different servlet.
  - e.g. search catalog, check out, confirm order etc.
- Client of the servlet can be any of the following
  - Browser
  - Applet
  - Java Application

Servlets

Servlet Lifecycle



Servlets

Servlet Communication

- Servlet can communicate with four different entities
  - Client during request/response cycle
  - With servlet container to get context/config information
  - With other resources on server e.g. servlets, EJBs
  - With external resources like databases, legacy systems, and EIS
- Client communication can be in many forms
- In Http communication
  - Request – Information parameters (as name value pairs)
  - Response
    - HTML (Browsers)
    - WML (Mobile Devices)
    - CSV (Spreadsheets)
    - XML (Communicating with non-java systems)
    - Serialized Objects

Servlets API

Servlets

Servlet API

- Contained in two packages
  - javax.servlet
  - javax.servlet.Http
- Contains 20 interfaces and 16 classes
  - Prevalence of interfaces allows servlet implementation to be customized to container

Servlets

JAVA Servlets

- Javax.servlet package can be extended for use with any application layer protocol
  - http is the most popularly used protocol
  - Javax.servlet.http package is extension of the javax.servlet package for http protocol
- The Servlet spec allows you to implement separate Java methods implementing each HTTP method in your subclass of HttpServlet.
  - Override the doGet() and/or doPost() method to provide normal servlet functionality.
  - Override doPut() or doDelete() if you want to implement these methods.
  - There's no need to override doOptions() or doTrace().
  - The superclass handles the HEAD method all on its own.

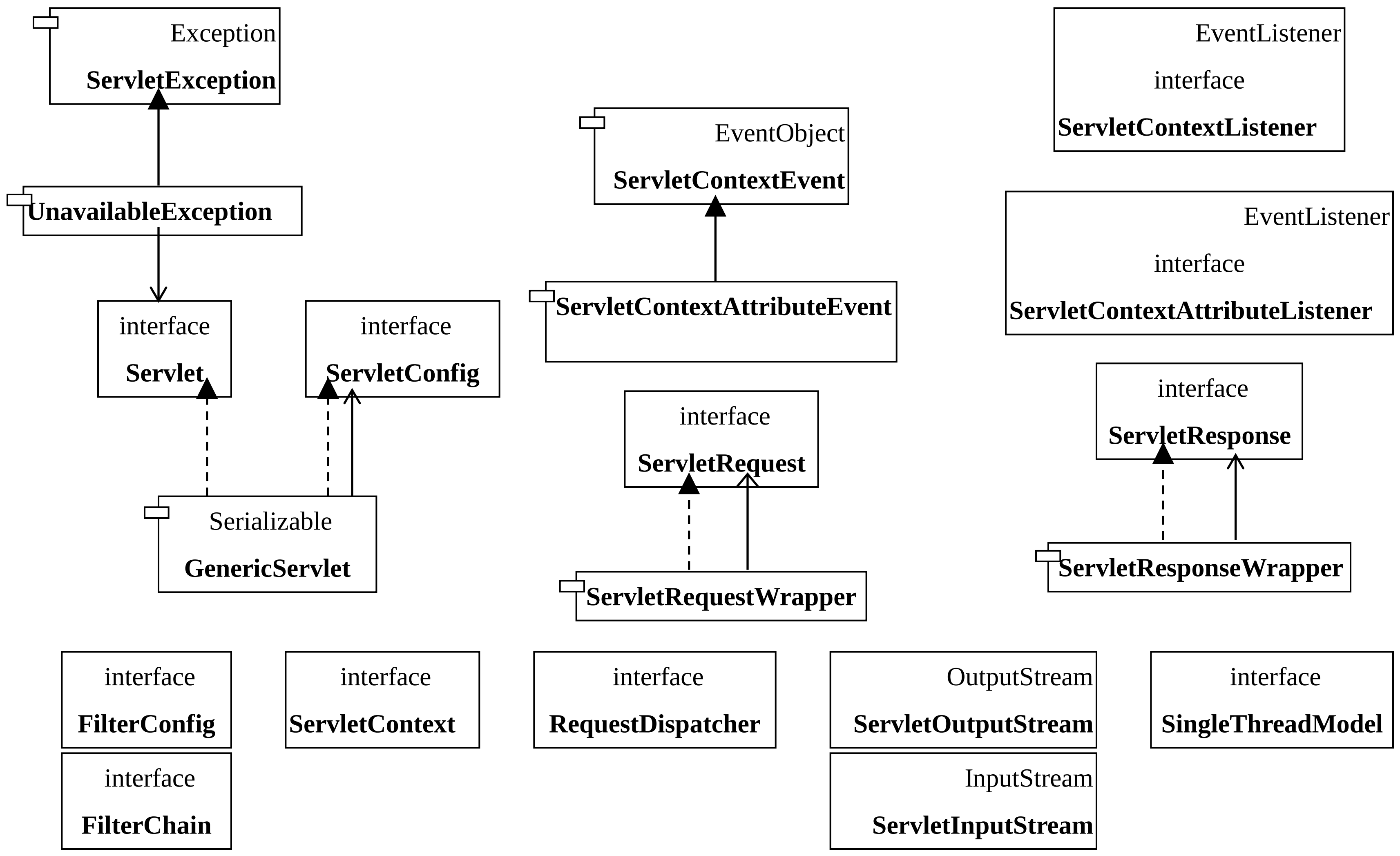
Servlets

Javax.servlet Package

- Provides the contract between the servlet/web application and the web container
- Used for creating protocol independent server applications
- Servlet interface defines the core of the entire package
  - Other interfaces provide additional services to the developer
- Contains 12 interfaces
  - 7 interfaces implemented by the package
  - 5 interfaces implemented by the user

Servlets

Class Diagram



Servlets

Interfaces

- Server implemented interfaces
  - ServletConfig
  - ServletContext
  - ServletRequest
  - ServletResponse
  - RequestDispatcher
  - FilterChain
  - FilterConfig
- User implemented interfaces
  - Servlet
  - ServletContextListener
  - ServletContextAttributeListener
  - SingleThreadModel
  - Filter

Servlets

Classes

- Servlet Classes
  - GenericServlet
  - ServletContextEvent
  - ServletContextAttriubuteEvent
  - ServletInputStream
  - ServletOutputStream
  - ServletRequestWrapper
  - ServletResponseWrapper
- Exception Classes
  - ServletException
  - UnavailableException

Servlets

Generic Servlet Class

- GenericServlet is abstract class that implements servlet interface
  - Requires implementing the service() method
  - Servlets normally extend from this class
- Methods
  - LifeCycle Methods
    - init()
    - service()
    - destroy()
  - Environment Methods
    - getServletContext()
    - getInitParameter(…)
    - getInitParameterNames()
  - Utility Methods
    - log(…)

Servlets
javax.servlet.http

- Javax.servlet package provides interfaces and classes to service client requests in protocol independent manner.
  - Javax.servlet.http package supports http-specific functions.
- Several of the classes are derived from the javax.servlet packaage
- Some methods from the javax.servlet package are also used
- Contains
  - 8 interfaces
  - 7 classes

Servlets

Classes and Interfaces

Interfaces

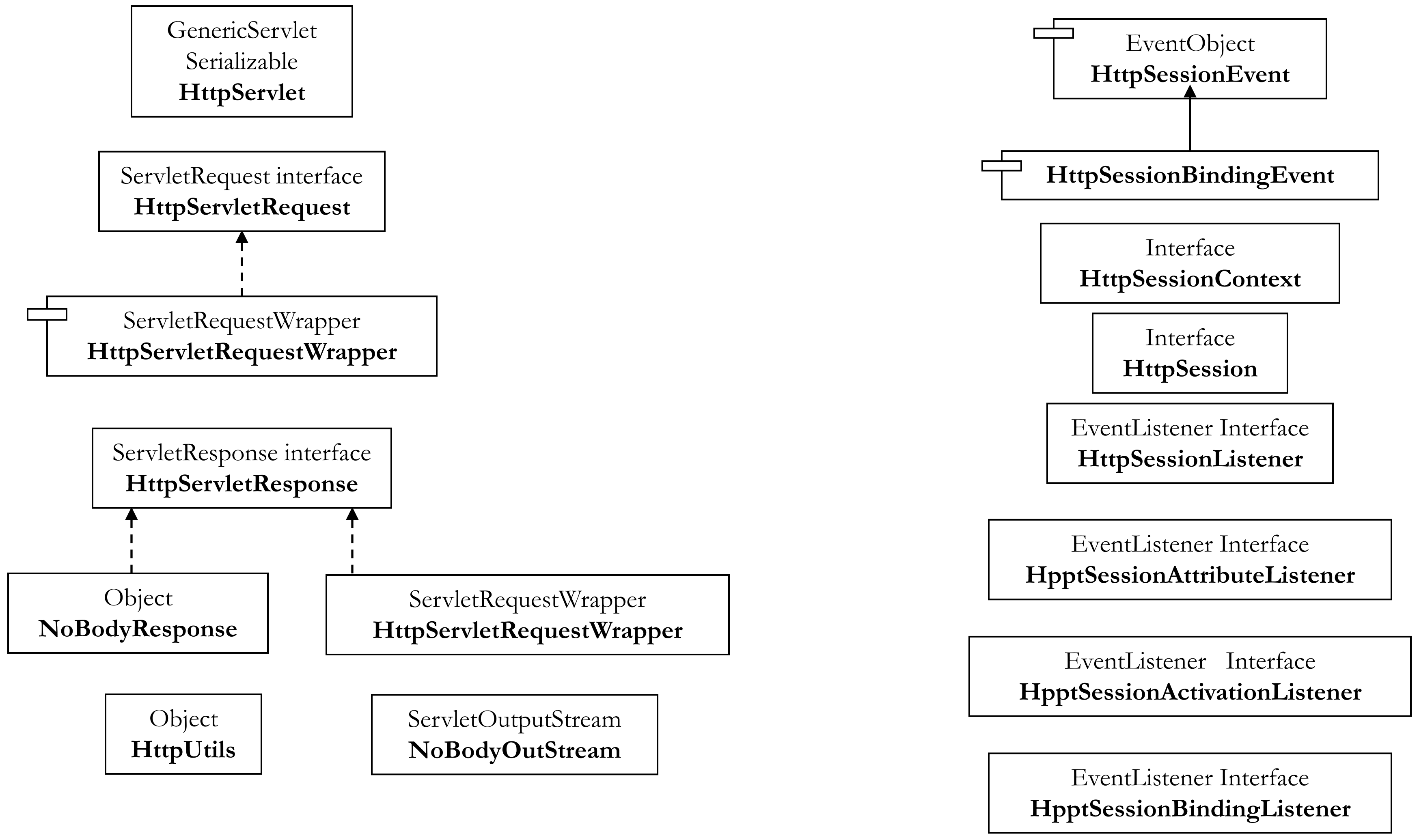
– HttpSession
– HttpServletRequest
– HttpServletResponse
– HttpSessionAttributeListener
– HttpSessionActivationListener
– HttpSessionBindingListener
– HttpSessionContext
– HttpSessionListener

Classes

– Cookie
– HttpServlet
– HttpServletRequestWrapper
– HttpServletResponseWrapper
– HttpSessionBindingEvent
– HttpSessionEvent
– HttpUtils

Servlets

Class Diagram



Servlets
HttpServlet Class

- Extends the Generic Servlet
  - Inherits the init() and destroy methods()
  - Overrides the service() method
- Service() method
  - Signature: Protected void service(HttpServletRequest req, HttpServletResponse res)
  - Forwards the request to the appropriate method
  - Developer should not normally override this method
- The developer needs to implement the methods corresponding to the request
  - doGet(), doPost(), doHead(), doPut()

Servlets

HttpServletRequest Interface

- Extends ServletRequest
- Inherited methods from ServletRequest
  - getParameterNames()
  - getParameter(String name)
  - getParameterValues(String name)
  - getServerName()
  - getServerPort()
  - getRequestDispatcher
- New methods defined
  - getCookies()
  - getHeader()
  - getPathInfo()
  - getContextPath()
  - getQueryString()

Servlets

HttpServletRequest Interface, cont'd.

- Extends ServletResponse
- Inherited methods from ServletResponse
  - getoutputStream()
  - getWriter(String name)
  - flushBuffer()
  - setContentType()
- New methods
  - encodeURL(String url)
  - encodeRedirectURL(String url)
- setDateHeader()
  - setStatus()
  - ....

Servlets

Cookie Class

- Constructor
  - Cookie (String name, String value)
- Methods
  - public void setMaxAge(int expiry)
  - public void setValue(String newValue)
- Can be added to the response by using
  - void addCookie(Cookie cookie) of HttpServletResponse
- Can be obtained from the request by using
  - Cookie[] getCookies() method of the HttpServletRequest

Servlets

Writing a Servlet

- Create a servletclass
  - extend HttpServlet
- Implement the doGet() or doPost() method
  - Both methods accept two parameters
    - HttpServletRequest
    - HttpServletResponse
  - Obtain parameters from HttpServletRequest Interface using
    - getParameter(String name)
  - Obtain the writer from the response object
  - Process input data and generate output (in html form) and write to the writer
  - Close the writer

Example 1

Example 1
Login Servlet

```
package edu.albany.mis.goel.servlets;
import javax.servlet.http.*;
import java.io.*;
public class Login extends HttpServlet {
 public void doPost(HttpServletRequest request, HttpServletResponse response) {
  // Get the parameter from the request
  String username = request.getParameter("username");
  // Send the response back to the user
  try {
   response.setContentType("text/html");
   PrintWriter writer = response.getWriter();
   writer.println("<html><body>");
   writer.println("Thank you, " + username + ". You are now logged into the system.");
   writer.println("</body></html>");
   writer.close();
  } catch (Exception e) {
   e.printStackTrace();
  }
 }
}
```

Example 1

Login.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
 <head>
  <title>Login</title>
 </head>
 <body>
  <h1>Login</h1>
  Please enter your username and password
  <form action="servlet/edu.albany.mis.goel.servlets.Login" method="POST">
   <p><input type="text" name="username" length="40">
   <p><input type="password" name="password" length="40">
   <p><input type="submit" value="Submit">
  </form>
 </body>
</html>
```

Example 1
web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>Login Servlet</display-name>
  <servlet>
    <servlet-name>Login</servlet-name>
    <servlet-class>edu.albany.mis.goel.servlets.Login</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Login</servlet-name>
    <url-pattern>/Login</url-pattern>
  </servlet-mapping>
</web-app>
```

Example 1
Login Deployment

- Compiling
  - Makefile contains all the scripts for compiling and deployment of the servlet
  - Needs to be modified for any give application
- Commands
  - make shutdown: shuts down the tomcat server
  - make clean: cleans up the current setup for the application
  - make all: compiles code, creates war file and deploys war file on server
  - make startup: starts the server again
- Running the servlet
  - http://localhost:8080/login/login.html

Example 2

Example 2
HttpRequestResponsServlet

```
package edu.albany.mis.goel.servlets;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import java.util.Date;
/**
 * Description:
 * @author Andrew Harbourne-Thomas
 * @version 1.0
 */
public class HttpRequestResponseServlet extends HttpServlet {
  private static int cookiesCreated = 0;
```

Example 2

Servlet – doGet()

```
/** Output a web page with HTTP request information and response data.
 *  @param request The object containing the client request
 *  @param response The object used to send the response back
 */
public void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException  {
   StringBuffer httpRequestTable = getHttpRequestTable(request);
   StringBuffer httpResponseTable = getHttpResponseTable(response);
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   //HTML page
   out.println("<html><head><title>RequestResponseServlet</title></head><body>");
   out.println("<h1>Request Information</h1>" + httpRequestTable + "<hr>");
   out.println("<h1>Response Information</h1>" + httpResponseTable);
   out.println("</body></html>");
   out.close();
 }
```

Example 2

HTMLTable Class

```
public class HTMLTable {
  private StringBuffer head;
  private StringBuffer rows;
  private StringBuffer foot;

  /** Initalises the StringBuffer Objects.
   */
  public HTMLTable() {
    head = new StringBuffer();
    head.append("<table width=\"90%\" align=\"center\">");
    head.append("<tr><th width=\"50%\" bgcolor=\"lightgrey\">Attribute</td>");
    head.append("<th width=\"50%\" bgcolor=\"lightgrey\">Value</td></tr>");
    rows = new StringBuffer();
    foot = new StringBuffer();
    foot.append("</table>");
  }
```

Example 2

HTMLTable Class, cont'd.

```
/** Appends the attribute and value in a row to the HTML table StringBuffer.
  * @param attribute The first column value.
  * @param value The second column value.
  */
 public void appendTitleRow(String attribute) {
   rows.append("<tr><td colspan=2><b><u>").append(attribute);
   rows.append("</u></b></td></tr>");
 }
 /** Appends the attribute and value in a row to the HTML table StringBuffer.
  *   @param attribute The first column value.
  *   @param value The second column value.
  */
 public void appendRow(String attribute, String value) {
   rows.append("<tr><td>").append(attribute);
   rows.append("</td><td><code>").append(value).append("</code></td></tr>");
 }
 /** Appends the attribute and value in a row to the HTML table StringBuffer.
  *   @param attribute The first column value.
  *   @param value The second column value.
  */
 public void appendRow(String attribute, int value) {
   appendRow(attribute, new Integer(value).toString());
 }
```

Example 2

HTMLTable Class, cont'd.

```
 /** Appends the attribute and value in a row to the HTML table StringBuffer
  *   @param attribute The first column value.
  *   @param value The second column value.
  */
 public void appendRow(String attribute, boolean value) {
   appendRow(attribute, new Boolean(value).toString());
 }
 /** Overrides Object.toString method to present a String representation of the HTML table built up.
  *   @return value The second column value.
  */
 public String toString() {
   return head.append(rows).append(foot).toString();
 }
 /** Presents a StringBuffer representation of the HTML table built up.
  *   @return value The second column value.
  */
 public StringBuffer toStringBuffer(){
   return head.append(rows).append(foot);
 }
}
```

Example 2

Servlet - getHttpRequestTable

```
/** Prepare a HTML table of information about the request made.
 *   @param request The object containing the client request
 *   @return String containing the table
 */
private StringBuffer getHttpRequestTable(HttpServletRequest request)  {
  HTMLTable table = new HTMLTable();
  table.appendRow("HTTP Request Method", request.getMethod());
  table.appendRow("Query String", request.getQueryString());
  table.appendRow("Context Path", request.getContextPath());
  table.appendRow("Servlet Path", request.getServletPath());

  //additional info if required
  /*
  table.appendRow("Path Info", request.getPathInfo());
  table.appendRow("Path Translated", request.getPathTranslated());
  table.appendRow("Request URI", request.getRequestURI());
  table.appendRow("Request URL", request.getRequestURL().toString());
  */
```

Example 2

Servlet – getHttpRequestTable, cont'd.

```
  // Get cookies from the user request
  Cookie[] ourCookies = request.getCookies();

  if (ourCookies == null || ourCookies.length == 0) {
    table.appendRow("Cookies", "NONE");
  } else  {
    for (int i = 0; i < ourCookies.length; i++) {
      String cookieName = ourCookies[i].getName();
      String cookieValue = ourCookies[i].getValue();
      table.appendRow("Cookie: <code>" + cookieName + "</code>", cookieValue);
    }
  }
  Enumeration e = request.getHeaderNames();
  while (e.hasMoreElements())  {
    String headerName = (String)e.nextElement();
    String headerValue = request.getHeader(headerName);
    table.appendRow("Header: <code>" + headerName + "</code>", headerValue);
  }
  return table.toStringBuffer();
}
```

Example 2

Servlet – getHttpRequestTable, cont'd.

```
/** Prepare a HTML table of information about the response made.
 *  @param response Gives access to the response object
 *  @return String containing the table
 */
 private StringBuffer getHttpResponseTable(HttpServletResponse response) {
   HTMLTable table = new HTMLTable();
   int cookieCount = cookiesCreated++;
   String name = Integer.toString(cookieCount);
   String value = new Date(System.currentTimeMillis()).toString();
   Cookie cookie = new Cookie(name, value);
   response.addCookie(cookie);
   table.appendRow("Cookie Added:<code>" + name + "</code>", value);
   return table.toStringBuffer();
 }
}
```

Tracking State

Tracking State

Cookies

- A Cookie is data (String) that the server passes to the browser and the browser stores on the server
  - Set of name value pairs
- Web servers place cookies on user machines with id to track the users
- Two types of cookies
  - Persistent cookies: Stored on hard drive in text format
  - Non-persistent cookies: Stored in memory and goes away after you reboot or turn off the machine

Tracking State

Cookie Attributes

- Attributes of a cookie
  - Name: Name of a cookie
  - Value: Value of the cookie
  - Comment: Text explaining purpose of cookie
  - Max-Age: Time in seconds after which the client should not send cookie back to server
  - Domain: Domain to which the cookie should be sent
  - Path: The path to which the cookie should be sent
  - Secure: Specifies if cookie should be sent via https
  - Version: Cookie version

    (0 – original Netscape version of Cookie

    1 – cookies standardized via RFC 2109)

Tracking State
Cookie Servlet

```
package edu.albany.mis.goel.servlets;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Random;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.Cookie;
import javax.servlet.ServletException;
public class CookieServlet extends HttpServlet
{
   protected void doGet(HttpServletRequest
      request,HttpServletResponse response)
      throws ServletException, IOException
   {
```

```
Cookie[] cookies = request.getCookies();
   Cookie token = null;
   if(cookies != null) {
     for(int i = 0; i < cookies.length; i++)
    {
      if(cookies[i].getName().equals("token"))
         {
           // Found a token cookie
           token = cookies[i];
           break;
      }
     }
   }
```

Tracking State

Cookies (Token)

```
response.setContentType("text/html");
PrintWriter writer = response.getWriter();
writer.println("<html><head><title>Tokens</title></head><body ");
writer.println("style=\"font-family:verdana;font-size:10pt\">");
String reset = request.getParameter("reset");
System.out.println("token = " + token);
if (token == null || (reset != null && reset.equals("yes"))) {
Random rand = new Random();
long id = rand.nextLong();
writer.println("<p>Welcome. A new token " + id + " is now established</p>");
// Set the cookie
token = new Cookie("token", Long.toString(id));
token.setComment("Token to identify user");
token.setMaxAge(-1);
token.setPath("/cookie/track");
```

Tracking State

Cookies, cont'd.

```
  response.addCookie(token);
  } else  {
     writer.println("Welcome back. Your token is " + token.getValue() +
        ".</p>"); }
   String requestURLSame = request.getRequestURL().toString();
   String requestURLNew = request.getRequestURL() + "?reset=yes";
   writer.println("<p>Click <a href=" + requestURLSame +
            ">here</a> again to continue browsing with the same identity.</p>");
   writer.println("<p>Otherwise, click <a href=" + requestURLNew +
            ">here</a> again to start browsing with a new identity.</p>");
   writer.println("</body></html>");
   writer.close();
 }
}
```

Tracking State

Cookies

```
package edu.albany.mis.goel.servlets;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Random;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.Cookie;
import javax.servlet.ServletException;
public class CookieServlet extends HttpServlet {
  protected void doGet(HttpServletRequest
                request,HttpServletResponse response)
    throws ServletException, IOException  {
    Cookie[] cookies = request.getCookies();
    Cookie token = null;
    if(cookies != null) {
      for(int i = 0; i < cookies.length; i++) {
          if(cookies[i].getName().equals("token"))  {
              // Found a token cookie
               token = cookies[i];
               break;
        }
      }
    }
response.setContentType("text/html");
PrintWriter writer = response.getWriter();
writer.println("<html><head><title>Tokens</title></head><body ");
writer.println("style=\"font-family:verdana;font-size:10pt\">");
String reset = request.getParameter("reset");
System.out.println("token = " + token);
if (token == null || (reset != null && reset.equals("yes"))) {
    Random rand = new Random();
    long id = rand.nextLong();
    writer.println("<p>Welcome. A new token " + id + " is now
established</p>");

    // Set the cookie
    token = new Cookie("token", Long.toString(id));
    token.setComment("Token to identify user");
    token.setMaxAge(-1);
    token.setPath("/cookie/track");
    response.addCookie(token);
}
else  {
    writer.println("Welcome back. Your token is " + token.getValue() +
".</p>");
}
```

Tracking State

URL Encoding

- http:// www.address.edu:1234/path/subdir/file.ext?query_string
  - Service → http
  - Host → www. Address. edu
  - Port → 1234
  - /path/subdur/file.ext → resource path on the server
  - query_string → additional information that can be passed to resource
- Http allows name value pairs to be passed to the resource
  - http:// www.test.edu/index.jsp?firstname=sanjay+lastname=goel
- The server can place the id of a customer along with the URL
  - http://www.fake.com/ordering/id=928932888329938.823948
- This number can be obtained by guessing or looking over some one's shoulder
  - Timeout for the sessions may be a few hours
  - User can masquerade as the owner of the id and transact on the web

Tracking State

URL Rewriting

```
package edu.albany.mis.goel.servlets;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Random;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
public class TokenServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request,HttpServletResponse
                response)
    throws ServletException, IOException {
    // Get the token from the request
    String tokenID = request.getParameter("tokenID");
   // Prepare for response
    response.setContentType("text/html");
    PrintWriter writer = response.getWriter();
writer.println("<html><head><title>Tokens</title></head><body ");
    writer.println("style=\"font-family:verdana;font-size:10pt\">");
}
```

```
if (tokenID == null) {
      // Client did not sent any token
      Random rand = new Random();
      tokenID = Long.toString(rand.nextLong());
      writer.println("<p>Welcome. A new token " + tokenID + " is now
                established</p>");
    }
    else {
    // Client sent the token back
      writer.println("<p>Welcome back. Your token is " + tokenID + ".</p>");
    //  Prepare links for sending requests back
    String requestURLSame = request.getRequestURL().toString() +
                "?tokenID=" + tokenID;
    String requestURLNew = request.getRequestURL().toString();
    // Write the response and close
    writer.println("<p>Click <a href=" + requestURLSame +
                        ">here</a> again to continue browsing with the same
                identity.</p>");
    writer.println("<p>Otherwise, click <a href=" + requestURLNew +
                        ">here</a> again to start browsing with a new
                identity.</p>");
    writer.println("</body></html>");
    writer.close();
  }
}
```

Tracking State
Hidden Form Fields

- HTML allows creation of hidden fields in the forms
- Developers use hidden fields to store information for their reference
- ID can be stored as a hidden form field
  - <Input Type=Hidden Name="Search" Value="key">
  - <Input Type=Hidden Name="id" Value="123429823">

Tracking State

Hidden Form Field

```
package edu.albany.mis.goel.servlets;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Random;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
public class HiddenFieldServlet extends HttpServlet {
  protected void doGet(HttpServletRequest
            request,HttpServletResponse response)
    throws ServletException, IOException {
    // Get the token from the request
    String token = request.getParameter("token");
    // Prepare for response
    response.setContentType("text/html");
    PrintWriter writer = response.getWriter();
    writer.println("<html><head><title>Tokens</title></head><body ");
    writer.println("style=\"font-family:verdana;font-size:10pt\">");
    if(token == null) {
      // Client did not sent any token
      Random rand = new Random();
      token = Long.toString(rand.nextLong());
      writer.println("<p>Welcome. A new token " + token + " is now
            established</p>");
    }
else {
    // Client sent the token back
      writer.println("<p>Welcome back. Your token is " + token + ".</p>");
    // Prepare a URL for sending requests back
    String requestURL = request.getRequestURL().toString();
    // Write a form with a hidden field
    writer.println("<p>");
    writer.println("<form method='GET' action='" + requestURL + "'>");
    writer.println("<input type='hidden' name='token' value='" + token + "'/>");
    writer.println("<input type='submit' value='Click Here'/>");
    writer.println("</form>");
    writer.println(" to continue browsing with the same identity.</p>");
    // Write another form without the hidden field
    writer.println("<p>");
    writer.println("<form method='GET' action='" + requestURL + "'>");
    writer.println("<input type='submit' value='Click Here'/>");
    writer.println("</form>");
    writer.println(" to start browsing with a new identity.</p>");
    writer.println("</body></html>");
    writer.close();
  }
 }
}
```

Tracking State
HttpSession Interface

- Provides methods to establish session between client and server
  - Session lasts for a specified time
  - Allows binding of objects over multiple requests
- Important Methods
  - getID()
  - getAttribute(String name)
  - getAttriubuteNames()
  - setAttribute(String name, Object value)
  - removeAttribute(String name)
  - inValidate()

Store

MainServlet

```
/** This is the main servlet of the application which reads the
 *  products from the product list and presents it to the user for
 *  selecting and addition to the shopping card. The data is read from
 *  an XML file and is added to a hashmap which is added to the
 *  ServletContext for future access.
 *  Steps:
 *  init()
 *  1. Gets the servletcontext
 *  2. Obtains the name of the product file from the context (init param)
 *  3. Creates a DOM parser
 *  4. Parses the product file and creates a document (xml data)
 *  5. Adds the product information to a Hashmap called product
 *  6. Adds the Hashmap to the context.
 *  doGetPost()
 *  1. Reads the products from the Hashmap
 *  2. Creates web page which contains standard header footer (dispatcher)
 *  3. Adds products to the web page and links them to the cartServlet
 */
package edu.albany.mis.goel.store;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

// JAXP packages
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.w3c.dom.*;
```

```
public class MainServlet extends HttpServlet {
  public void init() throws ServletException {
    //  Load the products from XML file provided by init parameter
    ServletContext context = getServletContext();
    InputStream productsFile = context.getResourceAsStream((String)
                 context.getInitParameter("productsFile"));
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    DocumentBuilder db = null;
    try {   db = dbf.newDocumentBuilder();
    }  catch (ParserConfigurationException pce) {
      throw new ServletException (pce.getMessage());
    }
    Document doc = null;
    try {  doc = db.parse(productsFile);
    }  catch (IOException ioe) {
      throw new ServletException(ioe.getMessage());
    } catch (SAXException se) {
      throw new ServletException(se.getMessage());  }
    NodeList productsList = doc.getElementsByTagName("product");
    HashMap products = new HashMap();
    Node product;
    for (int ctr = 0; ctr < productsList.getLength(); ctr ++ ) {
      product = productsList.item(ctr);
      NamedNodeMap attribs = product.getAttributes();
      Node attrib = attribs.getNamedItem("name");
      String name = attrib.getNodeValue();
      attrib = attribs.getNamedItem("price");
      String price = attrib.getNodeValue();
      Product p = new Product(ctr,name,price);
      products.put(new Integer(ctr),p);
    }
    //  Store products in the ServletContext
    context.setAttribute("products",products);
  }
```

Store

MainServlet

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    doGetOrPost(req,res);
  }
  public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    doGetOrPost(req,res);
  }
private void doGetOrPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    PrintWriter out = res.getWriter();
    // Include standard header
    RequestDispatcher dispatcher = req.getRequestDispatcher("/header.html");
    dispatcher.include(req,res);
    HashMap products = (HashMap) getServletContext().getAttribute("products");
    //  List the products, clickable to add to cart
    Iterator it = products.values().iterator();
    out.println("<table>");
    while (it.hasNext()) {
      out.println("<tr>");
      Product product = (Product) it.next();
      out.print("<td><a href='Cart?add=true&id=" + product.getId() +"'>");
      out.print(product.getName() + "</a></td><td>" + product.getPrice());
      out.println("</td>);
      out.println("</tr>");
    }
    out.println("</table>");
    // Include standard footer
    dispatcher = req.getRequestDispatcher("/footer.html");
    dispatcher.include(req,res);
  }
}
```

Store

Cart and Product

```
package edu.albany.mis.goel.store;
import java.util.*;
public class Cart {
  private HashMap items = new HashMap();
  // Default Cart Constructor
  public Cart() {
  }
  // Function to get items from the cart
  public Iterator getItems() {
    return items.values().iterator();
  }
  public void addItem(Product product) throws ItemAlreadyAddedException {
    Integer id = new Integer(product.getId());
    if (this.items.containsKey(id)) {
      throw new ItemAlreadyAddedException();
    }
    this.items.put(id, product);
  }
}

package edu.albany.mis.goel.store;
import javax.servlet.*;
public class ItemAlreadyAddedException extends ServletException {
}
```

```
package edu.albany.mis.goel.store;
public class Product {
  private String name;
  private String price;
  private int id;

  public Product(int id, String name, String price) {
    this.price = price;
    this.name = name;
    this.id=id;
  }

  public String getPrice() {
     return this.price;
  }

  public String getName() {
    return this.name;
  }

  public int getId() {
    return this.id;
  }

}
```

Store

CartServlet

```
package edu.albany.mis.goel.store;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class CartServlet extends HttpServlet {
 public void doPost(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException  {
 }
 public void doGet(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException {
  doGetOrPost(req,res);
 }
 private void doGetOrPost(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException {
  //  Get the cart if it exists
  HttpSession session = req.getSession();
  Cart cart = (Cart) session.getAttribute("cart");
  if (cart == null)  {
   cart = new Cart();
  }
  // Check to see if we are adding to the cart or we want to dispay the cart
  String adding = req.getParameter("add");
  PrintWriter out = res.getWriter();
  //  Add to it
  if (adding.equalsIgnoreCase("true"))  {
   addToCart(req, cart, out);
  }
  //  Display its contents
  displayCart(cart, out);
 }
```

```
private void addToCart(HttpServletRequest req, Cart cart, PrintWriter out)
  throws ItemAlreadyAddedException {
  //  Get the item to add from the request
    //  Get the products from the servletcontext
  HashMap products = (HashMap) getServletContext().getAttribute("products");
  //  Find the one represented by the ID that we passed in
  try {
   Integer id = new Integer(Integer.parseInt(req.getParameter("id")));
   Product p = (Product) products.get(id);
   //  Add it to the cart
   cart.addItem(p);
   //  add the cart to the session
   req.getSession().setAttribute("cart",cart);
   out.println("<b>Succesfully added product to cart!</b><br>");
  } catch (NumberFormatException nfe) {
   out.println("<b>Can't add product</b><br>");
  }
 }
 private void displayCart(Cart cart, PrintWriter out) {
  Iterator items = cart.getItems();
  out.println("<h1>Current Cart Contents:</h1>");
  out.println("<table>");
  while (items.hasNext()) {
   out.println("<tr>");
   Product p = (Product)items.next();
   out.println("<td>"+p.getName()+"</td>"+"<td>"+p.getPrice() +"</td>");
   out.println("<tr>");
  }
  out.println("</table>");
 }
}
```

Tracking State

CheckoutServlet

```
/** Checkout for the customer. This is also the place where the
 *  security check should be done to make sure that the customer is a
 *  registered customer. There are two ways of doing that. Currently
 *  security is not implemented
 *
 * 1. Declarative - Relies on the deployment
 * 2. Programmatic - Internally codes the security
 *
 * Steps
 * 1. Prints the contents of the shopping cart
 * 2. Asks the user to confirm his/her selection
 * 3. Sends the paget to the confirm page.
 */
package edu.albany.mis.goel.store;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.security.Principal;

public class CheckOutServlet extends HttpServlet {

  public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    doGetOrPost(req,res);
  }

  public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    doGetOrPost(req,res);
  }
private void doGetOrPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    // Get the writer
    PrintWriter out = res.getWriter();
   // include the cart display, and ask to confirm check out.
    System.out.println("Dispatching the request");
    RequestDispatcher dispatcher = req.getRequestDispatcher("/Cart?add=false");
    dispatcher.include(req,res);
    out.println("<br>Please Click Confirm to check out");
    out.println("<form action='confirmed.html'>" +
                         "<input type='submit' value='Confirm'></form>");
  }
}
```

Application Deployment

Application Deployment

Structure of Web Application

```
AppDir/
        index.html
        main.jsp
        images/
                company.jpg
                divider.jpg
        admin/
                admin.jsp
        WEB-INF/
                web.xml
                classes/edu/albany/mis/goel/servlets/
                        ShoppingCart.class
                        Catalog.class
                lib/
                        xereces.jar
                        xalan.jar
                edu/albany/mis/goel/servlets/
                        ShoppingCart.java
                        Catalog.java
```

- Public Resources that are downloaded directly to the client without processing
  - Lib files are standard libraries that the code may need
  - JSP files are an exception since they are converted to servlets and not downloaded directly
- Files which the web container processes but not client
  - Lib files are standard libraries that the code may need
- Source Files which are developed by the user
  - Package directory reduces chances of name conflicts

Application Deployment

Deployment of Web Applications

- Web applications are deployed in the web applications directory of the web server
  - In tomcat this directory is ${Tomcat_Home}/webapps
- Two separate ways of deploying web applications

Exploded Directory Format

- Development directory is copied to the application directory of the web server
- Used primarily in development mode when changes are frequent

Web Application Archive (WAR) Format

- Archived version of development directory is copied to application directory of web server
- Created using jar utility i.e. jar –cv0f SimpleWebApp.war .

Application Deployment

Deployment of Web Applications, cont'd.

- If web application is in a location different than the webapps directory context is defined
  - Location: ${Tomcat_Home}/conf/server.xml
- <context path="/store" docBase="/store.war" reloadable="true>
  - Context declares a context to exist with a base URL path of /store
  - The application can be accessed at http://localhost:8080/store/.
  - docBase tells tomcat where to find the web application
  - Relative path (/store.war) tells Tomcat that store.war is at the top level of the webapps directory
  - An absolute path can also be supplied I.e. c:/myapps/store.war
  - Reloadable set to true indicates that if the class or lib files change the application detects the change

Application Deployment

ServletContext

- Each application in a web container is associated with a context
  - All web resources are associated with the context.
- Servlet context is rooted at a known path within web container. (e.g. {Tomcat_Home}/webapps/store/home.html)
  - Context for this application is /store
  - User would access this as: http://localhost:8080/store/home.html
- There is a special object called servlet context.
  - A sandbox for the application (prevents name clashes and efficient downloading of classes without having to set classpath)
  - Allows servlets access container resources
  - Primary use of servlet context is to share attributes between servlets in an application.
- Context may be defined explicitly in a web server
  - Configuration Directory in Tomcat: ${Tomcat_Home}/conf/server.xml
  - <context path=“/examples” docBase=“examples” debug=“0” reloadable=“true”>

Application Deployment
Deployment Descriptor

- Conveys configuration information of a web application
- The primary elements of a deployment descriptor file
  - Servlet definitions & mappings
  - Servlet context initialization parameters
  - Error pages
  - Welcome pages
  - File based security
- Rules for the deployment descriptor file
  - Resides at the top level of the WEB-INF directory
  - Must be a well formed XML file called web.xml
  - Must conform to the dtd

    (located at http://java.sun.com/dtd/web-app-2-3.dtd)

Application Deployment

Deployment Descriptors - Header

- Header denotes the version of XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

- Describes the the DTD for the application

```
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  “http://java.sun.com/dtd/web-app_2_3.dtd">
```

- Description of the application enclosed in web-app tags

```
<web-app>
    Contents of the file
<web-app>
```

Application Deployment

Deployment Descriptors - Context

- Context parameters are parameters that are related to the entire application.
  - Any number of initialization parameters can be provided in the context
  - One initialization parameter for web application is shown below:

```
<context-param>
    <param-name>
      adminEmail
    </param-name>
    <param-vlaue>
     admin@wrox.com
    </param-value>
 </context-param>
```

- ServletContext object is used to obtain context information

e.g. String adminEmail = getServletContext().getInitParameter("adminEmail");

  - The methods in ServletContext are abstract, their implementations must be provided by the web container.

Application Deployment

Deployment Descriptors - Servlets

- Servlet Description, e.g.

```
<servlet>
  <servlet-name>storeservlet</servlet-name>
  <servlet-class>edu.albany.mis.goel.servlets.storeservlet<servlet-class>
  <init-param>
     <param-name>version<param-name>
    <param-value>0.1b<param-value>
  <init-param>
</servlet>
```

  - The above servlet is invoked by http://localhost:8080/store/home.html (Here store is the context of the application)
  - The initialization parameters are used for the specific servlet
  - They can be accessed using the ServletConfig object

  e.g. String version = getServletConfig().getInitParameter("version");

Application Deployment

Deployment Descriptors - Servlets

- Servlet mappings map servlets to specific URL pattern

  ```
  <servlet-mapping>
       <servlet-name>Servlet1</servlet-name>
       <url-pattern>/home.html<url-pattern>
   </servlet-mapping>
  ```

  – Allows web container to send requests to specific servlet
- Why is servlet mapping required?
  – A logical way to specify servlets would be to use context/servletname (i.e. http://localhost:8080/store/storeservlet)
  – Allows multiple urls to be mapped to same servlet
  – Allows implementation details to be hidden
- Servlets can be mapped to more than one URL thro the use of wildcards in <url-pattern>

  ```
  e.g. <servlet-mapping>
         <servlet-name>ValadatorServlet<servlet-name>
         <url-pattern>/*</url-pattern>
       </servlet-mapping>
  ```

  – The previous example maps every URL encountered to the same servlet

Application Deployment

Deployment Descriptors – Error Pages

- Error pages allow the application to specify pages to be shown when particular errors occur
  - Used for Java Exceptions and Http Errors.
  - The error page shown below is displayed when the server encounters a java.lang.ArithmeticException.

```
<error-page>
    <exception-type> java.lang.ArithmeticExceception </exception-type>   ← Exception Type
     <location>/error.html</location>                                     ← Resource to Show
 </error-page>
```

  - The error page shown below is displayed when the server encounters a an Http error

```
<error-page>
    <error-code>404</error-code>                      ← Http Error Code
     <location>/404.html</location>                   ← Resource to Show
 </error-page>
```

Application Deployment

Deployment Descriptors - Miscellaneous

- Application Name & Description

```
<web-app>
   <display-name> Music Store</display-name>
   <description>Application for Music Rentals</description>
</web-app>
```

- Welcome Pages

```
<welcome-file-list>
    <welcome-file>index.html</welcome-file>          ← Welcome File URL
 </welcome-file-list>
```

Application Deployment

Security Constraints

- Define Security Constraint (resource collection & authorization constraint)

```
<security-constraint>
        <web-resource-collection>
                    <web-resource-name>CheckOutResource</web-resource-name>
                     <url-pattern>/CheckOutServlet/*</url-pattern>
                    <http-method>GET</http-method>
                     <http-method>POST</http-method>
        </web-resource-collection>
        <auth-constraint>
                    <role-name>storeuser</role-name>                          ← Welcome File URL
        </auth-constraint>
</security-constraint>
```

- Define Login Configuration

```
<login-config>
        <auth-method>FORM</auth-method>
        <realm-name>Wrox Store Checkout</realm-name>
        <form-login-config>
                    <form-login-page>/login.html</form-login-page>
                     <form-error-page>/error.html</form-error-page>
        </form-login-config>
</login-config>
```

- Define Users in Tomcat (Add users in ${Tomcat_Home}/conf/tomcat-users.xml)

```
<tomcat-users>
     <user name="tomcat" password="tomcat" roles="tomcat" />
     <user name="role1" password="tomcat" roles="role1" />
</tomcat-users>
```

Application Deployment
ServletConfig Interface

- ServletConfig Object is used to pass initialization parameters to a servlet
- Useful methods
  - getServletName(): Returns name of servlet
  - getServletContext(): Returns servletContext object
  - getInitParameter(String name): returns value of the specified parameter (null if not present)
  - getInitParameterNames(): Gets names of all the parameters in the initialization list.

Application Deployment
ServletContext Interface

- ServletContext is specific to a particular web application running in a JVM
  - Each web application in a container will have a single servlet context associated with it.
  - Allows you to maintain state across all servlets and clients in the application
  - Also acts a shared repository for common attributes to all servlets
  - Allows servlets to share data with each other
- ServletContext Object also used for communication with host server
  - Allows servlet to get information about server on which it is running
- A typical use of this would be in a chat application

Application Deployment
ServletContext Interface, cont'd.

- Methods
  - getContext(String uripath)
  - getMimeType()
  - getResourcePaths()
  - getRequestDispatcher()
  - getRealPath()
  - getServerInfo()
  - getInitParameter()
  - getAttribute()
  - setAttribute()
  - ...

Session Management

Session Management

Basics

- HTTP is a stateless protocol. Each re.quest and response stand alone
- Without session management, each time a client makes a request to a server, it's brand new user with a brand new request from the server's point of view.
- A session refers to the entire interaction between between a client and a server from the time of the client's first request, which generally begins the session, to the time the session is terminated.

Session Management

Creating and Using Sessions

- Two methods of the HttpServletRequest object are used to create a session:
  - HttpSession getSession( );
  - HttpSession getSession(boolean);
- Other methods for dealing with sessions:

Method	**Description**
String getRequestedSessionID( )	Gets the ID assigned by the server to the session
Boolean isRequestSessionIdValid( )	Returns true if the request contains a valid session ID
Boolean isRequestSessionIdFromCookie( )	Returns true if the session ID was sent as part of a cookie
Boolean isRequestSessionIdFromURL( )	Returns true if the session ID was sent through URL rewriting

Session Management

What do you do with a session?

- Sessions are useful for persisting information about a client and a client’s interactions with an application.
- To do that, the HttpSession interface defines a number of mehods:
  - setAttribute(String, Object)
  - getAttribute(String)

Forwarding and Including Requests

Forwarding and Including Requests

Obtaining RequestDispatcher

- From ServletRequest
  - RequestDispatcher getRequestDispatcher(String path)
  - The path argument can be a relative path or absolute path
  - If the path is absolute relative to application context it starts with a "/" e.g. /Login
  - If the path if relative it is interpreted relative to the current web component location, e.g. if web component is /store then case would be considered /store/case
- From ServletContext
  - ServletContext getServletContext()
  - RequestDispatcher getNamedDispatcher(String name)
  - RequestDispatcher getRequestDispatcher(String path)
  - The path argument should always start with a / and is interpreted relative to the application context

Forwarding and Including Requests

Using RequestDispatcher

- Forwarding Request
  - void forward(ServletRequest request, ServletResponse response) throws ServletException, java.io.IOException
  - Calling servlet should not write any data to the response stream before calling this method
  - If response data is sent to the stream before calling forward an error is thrown
- Including Resource
  - void include(ServletRequest req, ServletResponse res) throws ServletException, java.io.Exception
  - You can safely write to the ResponseStream before calling the include function.

Forwarding and Including Requests

Adding Parameters

- Parameters are added for use in the forwarded request
- Several methods defined in ServletRequest Interface
  - Object getAttrubute(String name)
  - Enumeration getAttributeNames()
  - void setAttribute(String name, Object o)
  - void removeAttribute(String name)
- The calling servlet can set the attributes
- The receiving servlet will use getAttribute(String) method to retrieve the values