Servlets

- Objectives
  - Introduce the Java EE platform
  - Present the Servlets API

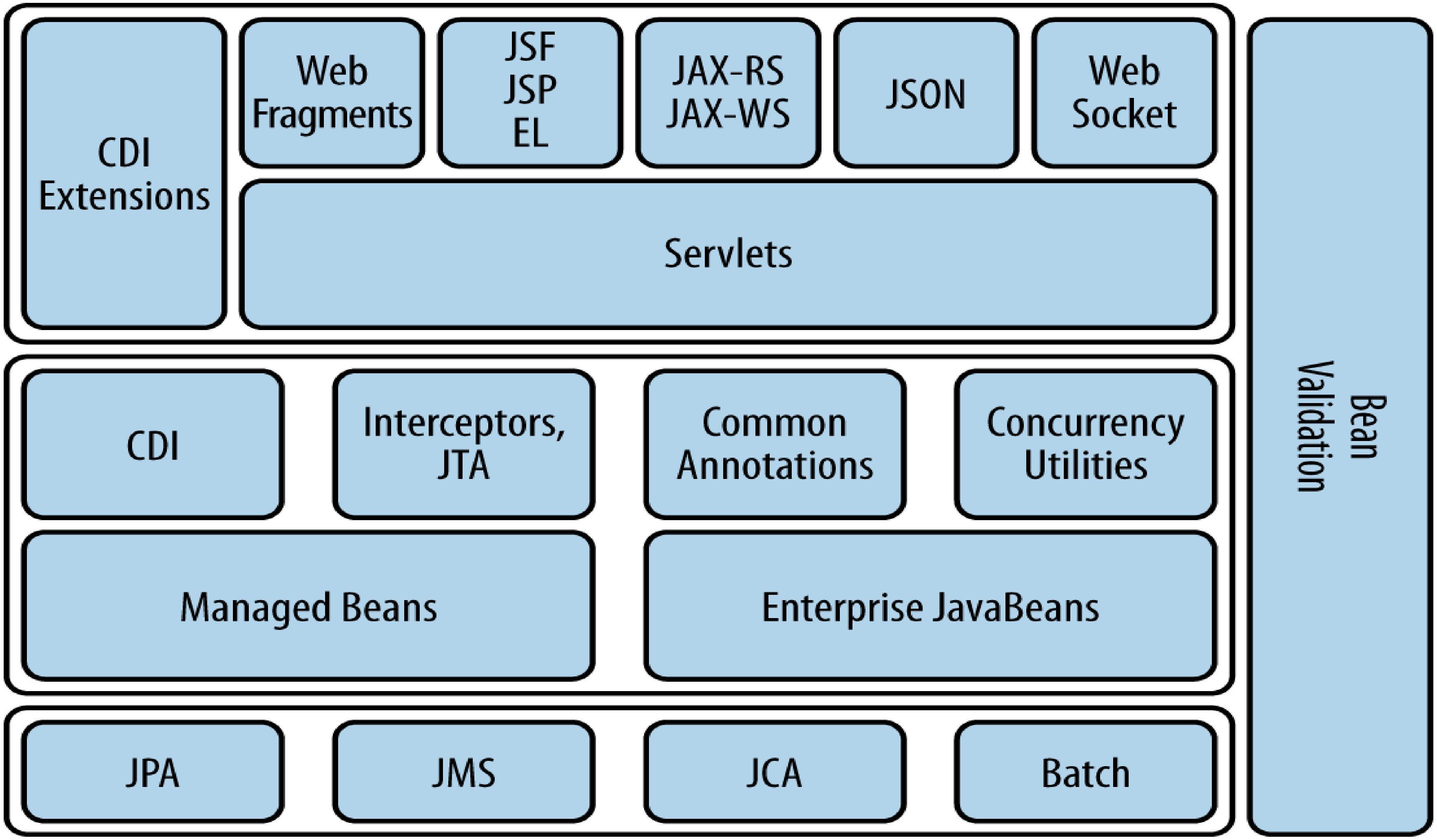
Part1 : Introduction

- Java EE Platform
- Introduction to Servlets
- Servlet Definition
- Servlet Life Cycle

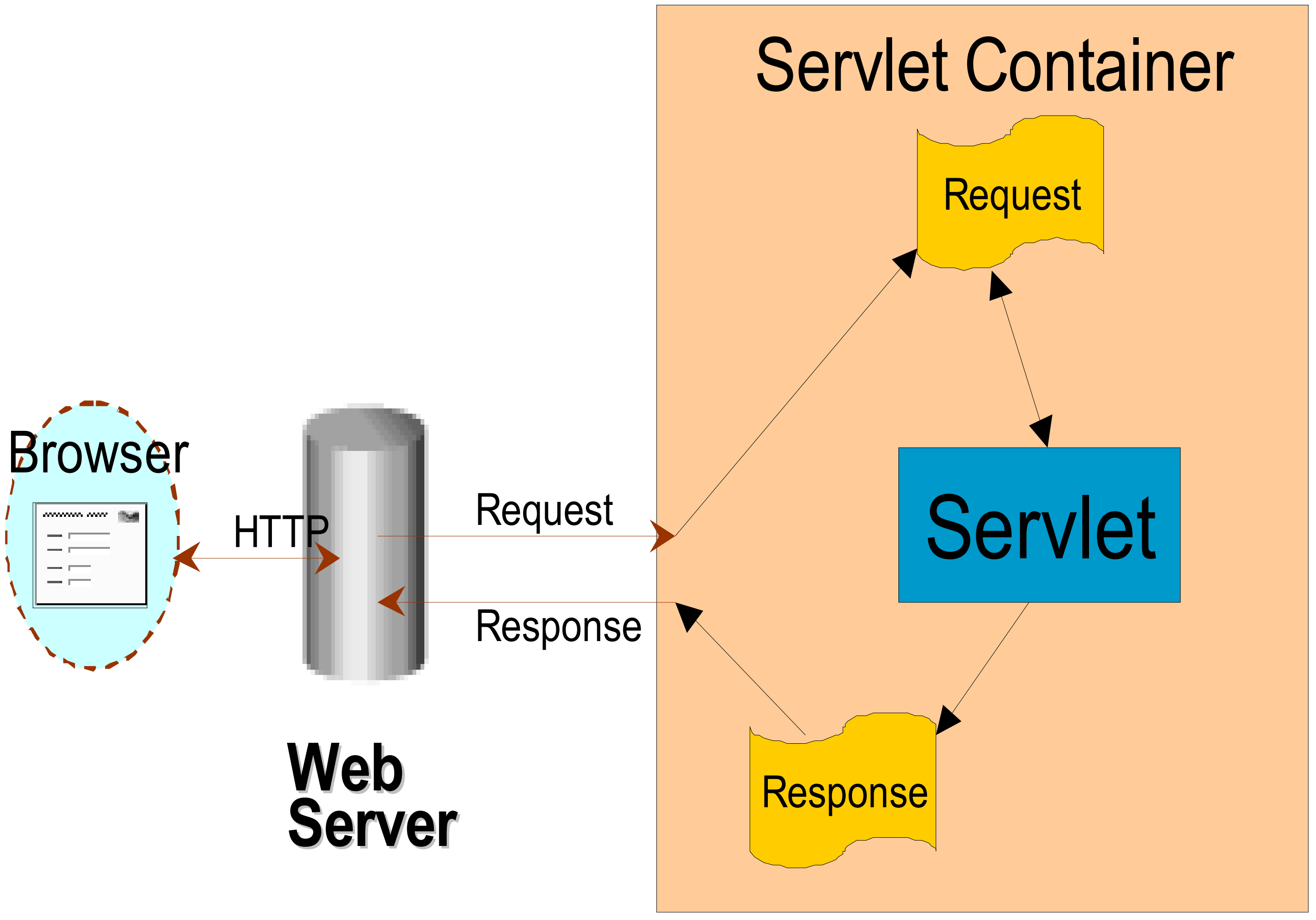
Java Platform, Enterprise Edition (Java EE)

- Formerly J2EE
- Developed by SUN – now own by Oracle
  - open standard
- Offers a suite of software specification to design, develop, assemble and deploy enterprise applications
  - n-tier, Web-enabled, server-centric, component-based
- Provides a distributed, component-based, loosely coupled, reliable and secure, platform independent application environment.

Java EE Technologies



Servlets

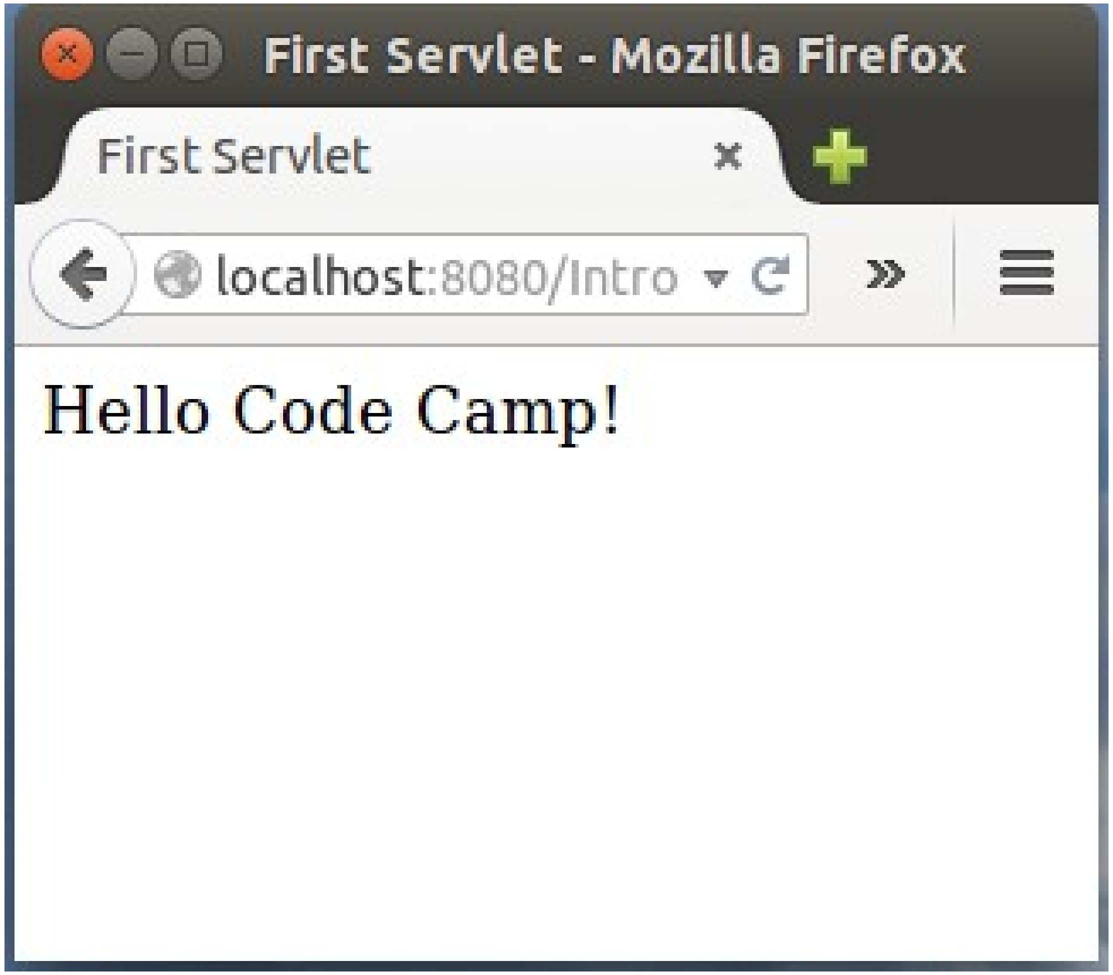

Servlet Container
Request
Browser
HTTP
Request
Servlet
Response
Response
Web Server

Example of servlet

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

@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
     HttpServletResponse response)
         throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<title>First Servlet</title>");
    out.println("<big>Hello Code Camp!</big>");
  }
}
```

Example of servlet



Hello Code Camp!

WebServlet Definition

```
@WebServlet("/account")
public class AccountServlet extends javax.servlet.http.HttpServlet
{

    //. . .

}
```

- Plain Old Java Object (POJO)
  - @WebServlet annotation – specifies *URL Pattern*
  - extends javax.servlet.http.HttpServlet

WebServlet Definition

```
@WebServlet(urlPatterns="/account",
    initParams={
        @WebInitParam(name="type", value="checking")
    }
)
public class AccountServlet extends javax.servlet.http.HttpServlet {
    String type = null;
    @Override
    public void init(ServletConfig config) throws ServletException {
        type = config.getInitParameter("type");
        //. . .
    }
}
```

- @WebInitParam used to specify initialization parameter
  - retrieved in init method

Deployment Descriptor

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
<servlet>
    <servlet-name>AccountServlet</servlet-name>
    <servlet-class>org.sample.AccountServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>AccountServlet</servlet-name>
    <url-pattern>/account</url-pattern>
</servlet-mapping>
</web-app>
```

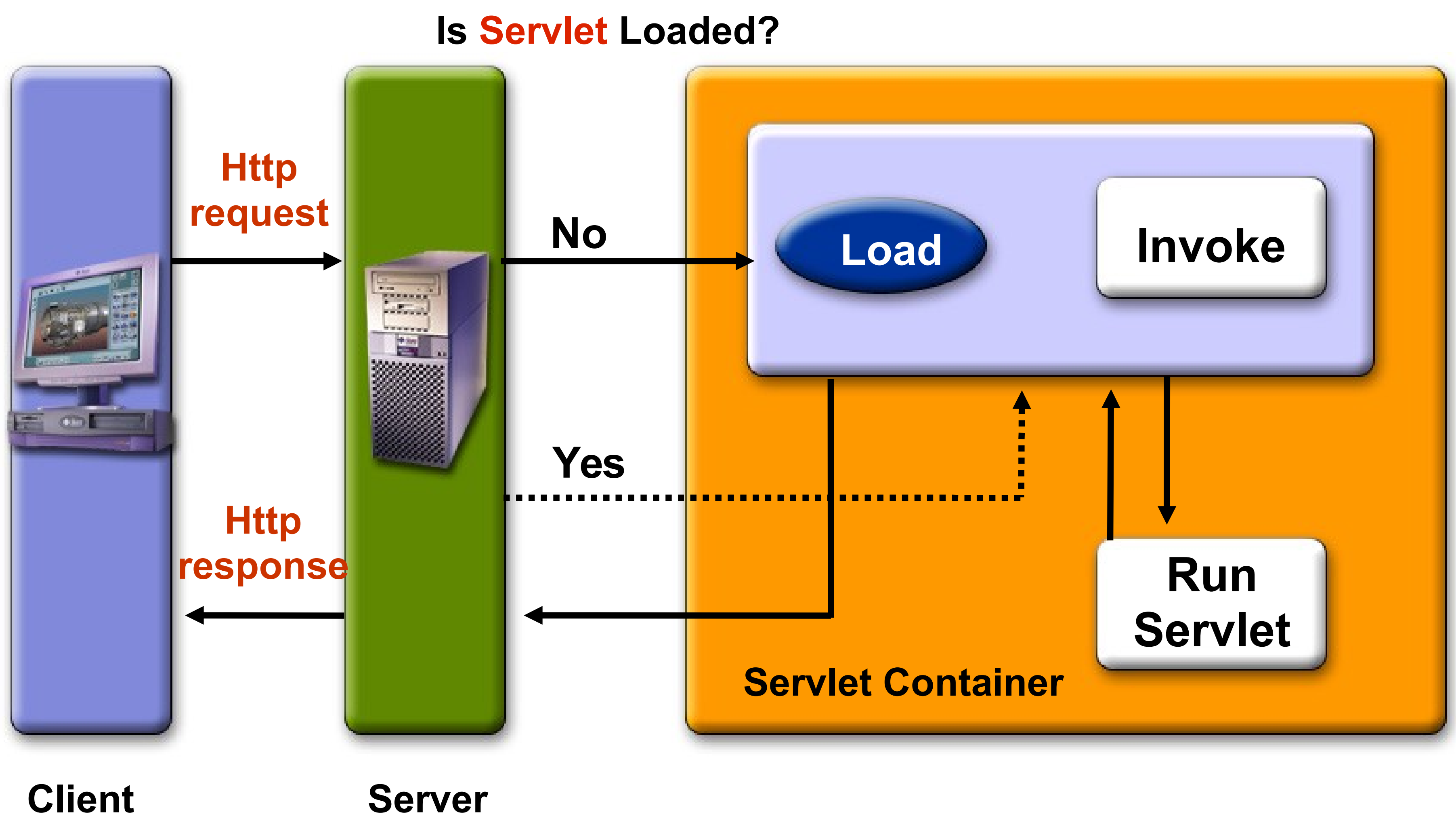
- *web.xml*
  - Can be used as alternative to @*WebServlet* annotation

Servlet Containers

- Servlet runs in a “Servlet Container”

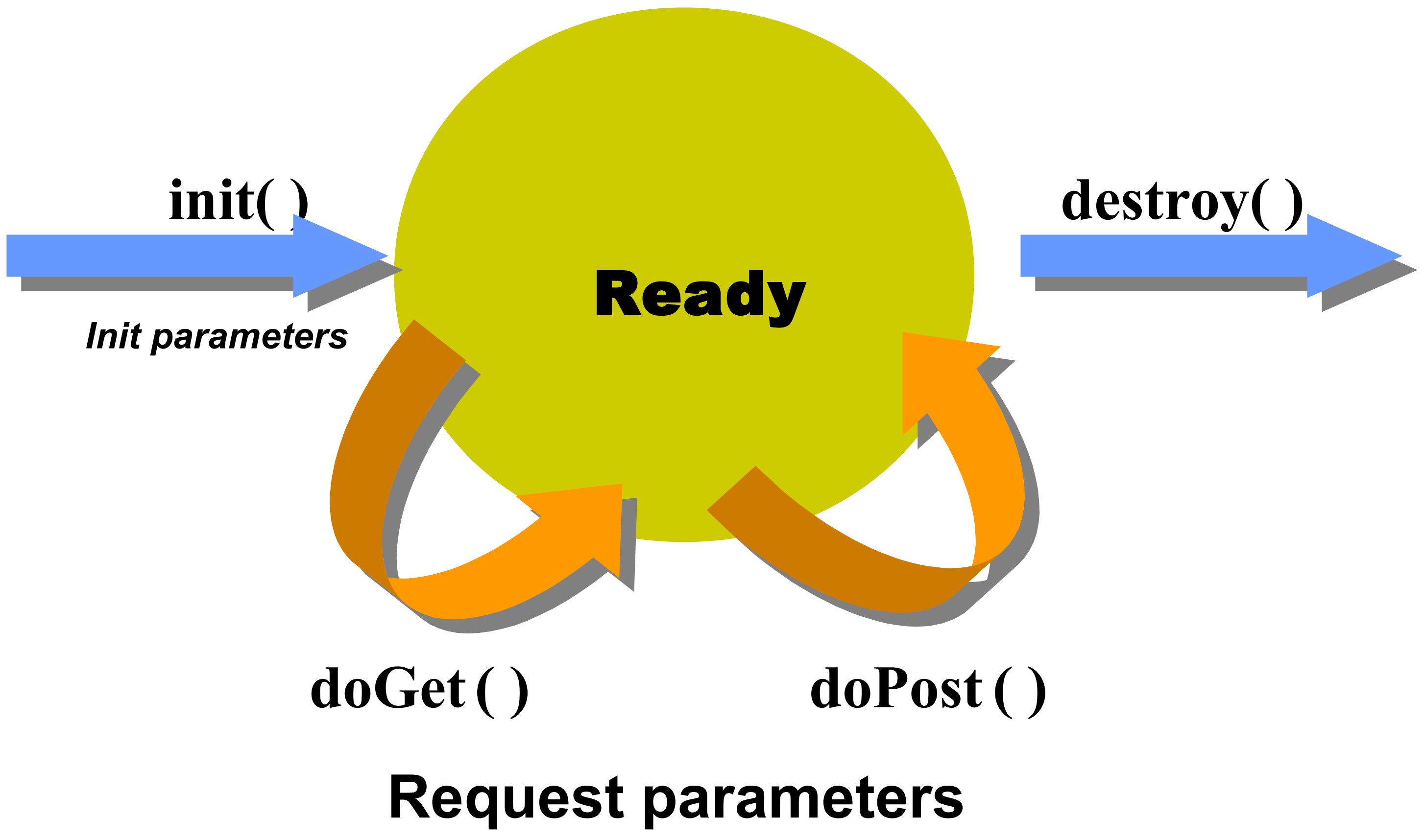
- Several existing “Servlet Containers”
  - all need to satisfy a servlet container specification
  - examples: Apache Tomcat, Jetty, Redhat JBoss, BEA Weblogic, Oracle Glassfish, IBM Websphere, ...

Servlet Life Cycle



Servlet Life-Cycle methods

- Methods invoked by Container
- defined in javax.servlet.http.HttpServlet and super classes
- Programming a servlet consists of overriding these methods



Servlet Life-Cycle methods

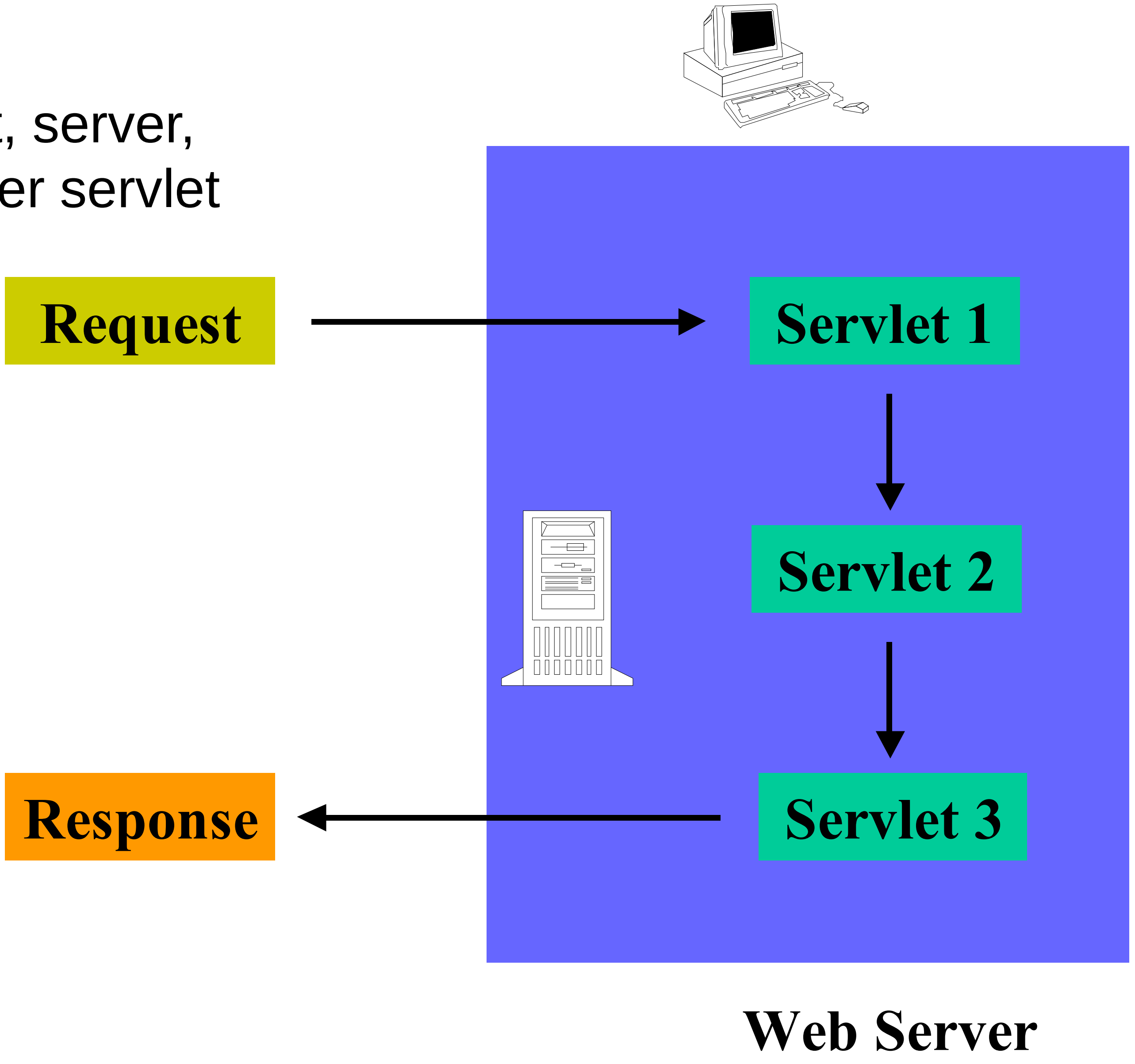
- init()
  - Invoked once when the servlet is first instantiated
  - Perform any set-up in this method
- destroy()
  - Invoked before servlet instance is removed
  - Perform any clean-up
    - Closing a previously created database connection
- doGet(), doPost(), doXxx()
  - Handles HTTP GET, POST, etc. requests
  - Override these methods in your servlet to provide desired behavior

Part2 : Requests

- HTTP Request
- Handling Form Data
- Multipart Requests
- Getting Request Information

Requests

data,
client, server,
header servlet
itself



HTTP Request URL

- Contains the following parts
  - http://[host]:[port]/[request path]?[query string]
- [request path] is made of
  - Context: /<context of web app>
  - Servlet name: /<component alias>
  - Path information: the rest of it
- Examples
  - http://localhost:8080/hello1/greeting
  - http://localhost:8080/hello1/greeting.jsp
  - http://daydreamer/catalog/lawn/index.html

HTTP Requests

```
@WebServlet("/account")
public class AccountServlet extends javax.servlet.http.HttpServlet {
    @Override
    protected void doGet( HttpServletRequest request,
                          HttpServletResponse response) {
        //. . .
    }
}
```

- Servlet implements *doXXX* method to handle HTTP requests
  - Get (doGet)
  - Post (doPost)
  - ...

HTTP Requests

- GET requests:
  - User entered information is appended to the URL in a query string
  - Can only send limited amount of data
    - .../servlet/ViewCourse?FirstName=John&LastName=Wayne
- POST requests:
  - User entered information is sent as data (not appended to URL)
  - Can send any amount of data

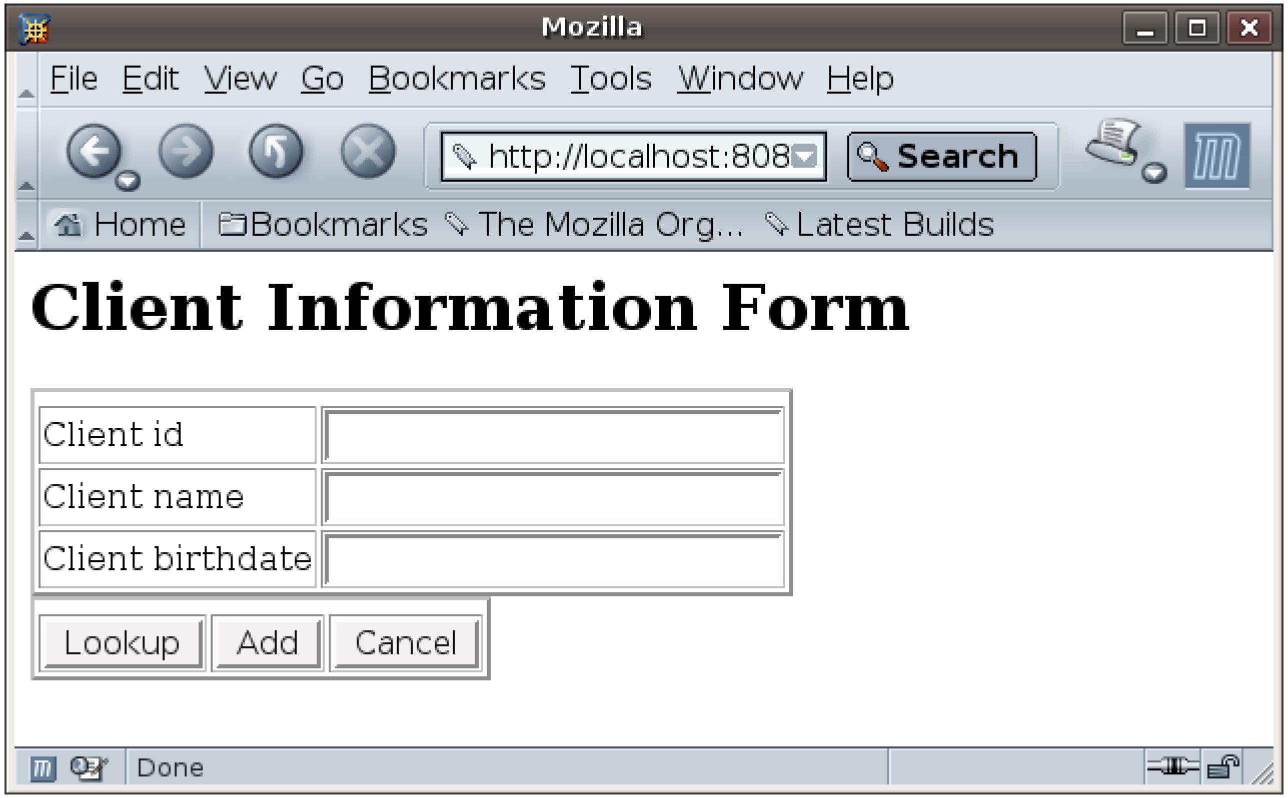
HTTP Request

```
protected void doGet(HttpServletRequest request,
                     HttpServletResponse response) {
    String txValue = request.getParameter("tx");
    //. . .
}
```

- HttpServletRequest object includes
  - request parameters
  - HTTP headers
  - HTTP cookies
  - URL info (Path, Host, Port, Context)
  - ...

HTTP Requests

- Forms are used to submit request through web pages
  - data is send using GET or POST


Mozilla
File Edit View Go Bookmarks Tools Window Help
http://localhost:808
Search
Home Bookmarks The Mozilla Org... Latest Builds
Done

Client Information Form

Client id	
Client name	
Client birthdate	

Lookup Add Cancel

  - data received as request parameters by Servlets

Handling Form Data

```
<html>
  <body>
      <h1>Client Information Form</h1>
      <form name="userForm" action="lookup" method="POST">
        Client id <input type="text" name="id" value=""/>
        Client name <input type="text" name="name" value="" />
        Client birthdate <input type="text" name="birthdate" value="" />
        <input type="submit" value="Lookup" name="lookup" />
        <input type="submit" value="Add" name="add" />
        <input type="reset" value="Cancel" name="cancel" />
      </form>
  </body>
</html>
```

```
@WebServlet("/lookup")
public class LookupServlet extends HttpServlet {
  protected void doPost(HttpServletRequest request,
                                    HttpServletResponse response)
    throws ServletException, IOException {
        ...
       // get value of id field
       String id = request.getParameter("id");
      // get value of name field
       String name = request.getParameter("name");
      // get value of birthdate file
       String birthdate = request.getParameter("birthdate");
      ...
      // check if lookup is pressed
      if (request.getParameter("lookup") != null) {
         // do whatever should be done
      }
      // check if add is pressed
      if (request.getParameter("add") != null) {
         // do whatever should be done
      }
  }
}
```

Handling Multipart Requests

```
@WebServlet(urlPatterns = {"/FileUploadServlet"})
@MultipartConfig(location="/tmp")
public class FileUploadServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
                  throws ServletException, IOException {
        for (Part part : request.getParts()) {
            part.write("myFile");
        }
    }
}
```

- *@MultipartConfig* indicates that servlet expects a request of type *multipart/form-data*.

Getting Client & Server Information

- Servlet can get client information from the request
  - String request.getRemoteAddr()
    - Get client's IP address
  - String request.getRemoteHost()
    - Get client's host name
- Servlet can get server's information:
  - String request.getServerName()
    - e.g. "www.sun.com"
  - int request.getServerPort()
    - e.g. Port number "8080"

Getting Misc. Information

- Input stream
  - ServletInputStream request.getInputStream()
  - java.io.BufferedReader request.getReader()
- Protocol
  - java.lang.String request.getProtocol()
- Content type
  - java.lang.String request.getContentType()
- Is secure or not (if it is HTTPS or not)
  - boolean request.isSecure()

Context, Path, Query, Parameter Methods

- String getContextPath()
- String getQueryString()
- String getPathInfo()
- String getPathTranslated()

Example – adding a link to a Servlet in a Servlet

- Suppose you want to add a link such that a servlet bound at context */SecondServlet* is called when clicked

```
...
// to make sure that full URL path is constructed
String Url = request.getContextPath() + “/SecondServlet”;
// use for session tracking
String encodedUrl = response.encodeURL(Url);
out.println(“<A HREF=\”” + encodedUrl + “\””>Link to SecondServlet</A>”);
...
```

Cookie Method (in HTTPServletRequest)

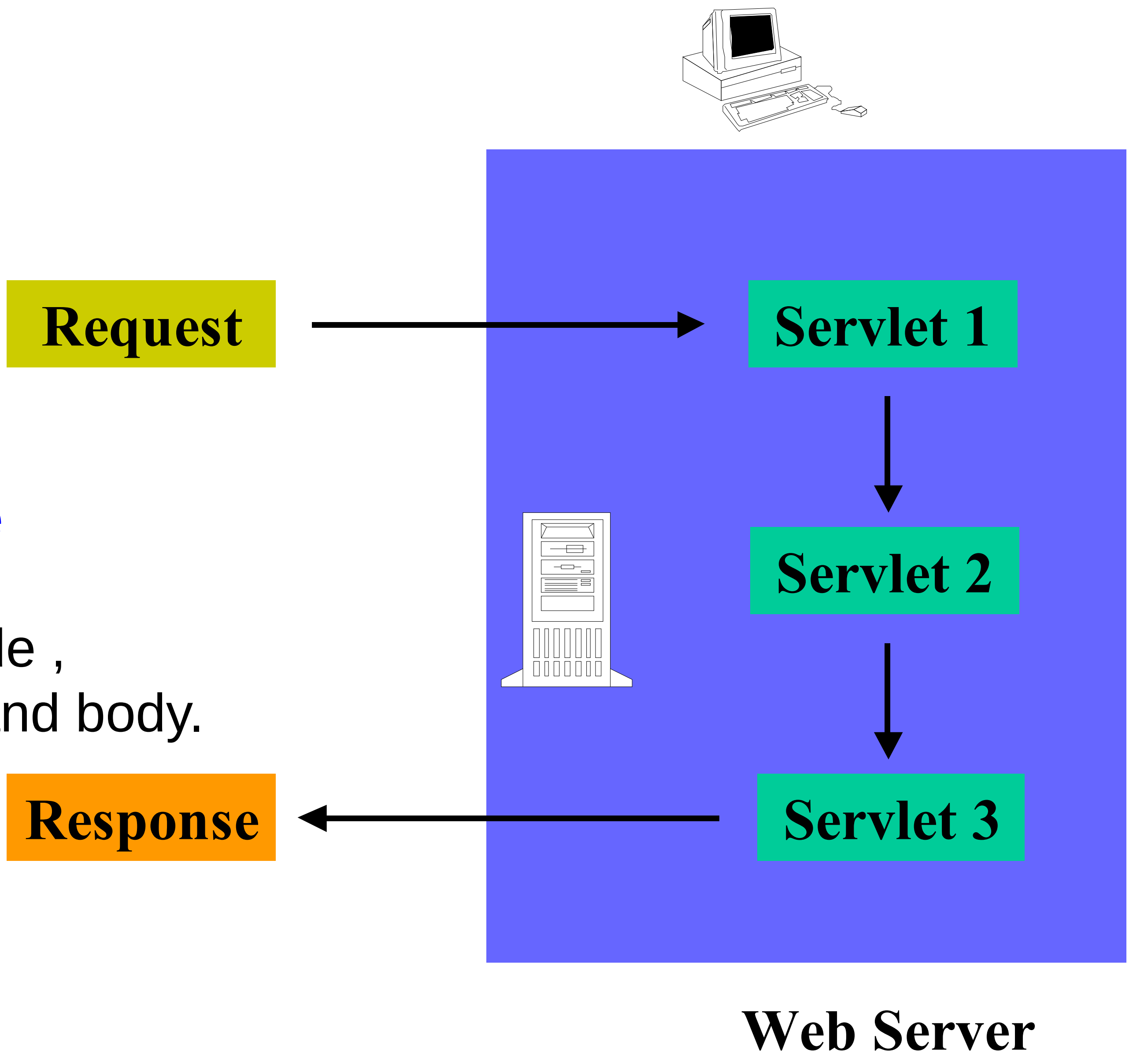
- Cookie[] getCookies()
  - an array containing all of the Cookie objects the client sent with this request
  - a particular cookie is found by looking-up in the array

```
String cookieName = “..”;
Cookie[] cookies = request.getCookies();
 if (cookies != null) {
   for(int i=0; i<cookies.length; i++) {
     Cookie c = cookies[i];
     if ((c.getName().equals(cookieName))  {
       doSomethingWith(cookie.getValue());
       break;
     }
   }
 }
```

Part3 : Responses

- Writing Response Body
- Setting Response Status
- Handling Errors

Responses



Response Structure:
status code ,
headers and body.

HTTP Response

```
protected void doGet(HttpServletRequest request,
                     HttpServletResponse response) {
    try (PrintWriter out = response.getWriter()) {
        out.println("<html><head>");
        out.println("<title>MyServlet</title>");
        out.println("</head><body>");
        out.println("<h1>My First Servlet</h1>");
        //. . .
        out.println("</body></html>");
    } finally {
    //. . .
    }
}
```

- Servlet creates response sent back to Client

Writing a Response Body

- Response body could either be a PrintWriter or a ServletOutputStream
- PrintWriter
  - Using response.getWriter()
  - For character-based output
- ServletOutputStream
  - Using response.getOutputStream()
  - For binary (image) data

Servlet Response (HttpServletResponse)

- Contains data passed from servlet to client
- Allows methods to
  - Retrieve an output stream
  - Indicate content type
  - Indicate whether to buffer output
  - Set localization information
  - Set HTTP response status code
  - Set Cookies

Methods for Setting HTTP Response Status Codes

- public void setStatus(int statusCode)
  - Status codes are defined in HttpServletResponse
  - Status codes are numeric fall into five general categories:
    - 100-199 Informational
    - 200-299 Successful
    - 300-399 Redirection
    - 400-499 Incomplete
    - 500-599 Server Error
  - Default status code is 200 (OK)

Example of HTTP Response Status

```
HTTP/ 1.1 200 OK
Content-Type: text/ html
<! DOCTYPE ...>
<HTML
...
</ HTML>
```

Methods for Sending Error

- Error status codes (400-599) can be used in sendError methods.
- public void sendError(int sc)
  - The server may give the error special treatment
- public void sendError(int code, String message)
  - Wraps message inside small HTML document

setStatus() & sendError()

```
try {
  returnAFile(fileName, out)
}
catch (FileNotFoundException e)
    {  response.setStatus(response.SC_NOT_FOUND);
       out.println("Response body");
    }

  has same effect as

try {
  returnAFile(fileName, out)
}
catch (FileNotFoundException e)
    {   response.sendError(response.SC_NOT_FOUND,"Response body"); }
```

Handling Errors

- Web container generates default error page
  - Possible to specify custom error pages
- To handle errors
  - Create appropriate error pages for error conditions
  - Modify the ***web.xml*** accordingly

Handling Errors

```
<error-page>
    <error-code>404</error-code>
    <location>/error-404.html</location>
</error-page>

<error-page>
    <exception-type>org.example.MyException</exception-type>
    <location>/error.html</location>
</error-page>
```

Part4 : Scope

- Scope of Objects
- The Web Context
- Including Ressources
- Forwarding

Scope of objects

- Enables sharing information among collaborating web components via attributes maintained in Scope objects
  - Attributes are name/object pairs
- Attributes maintained in the Scope objects are accessed with
  - getAttribute() & setAttribute()
- 4 Scope objects are defined
  - Web context, session, request, page

Scope of objects

- Web context (ServletContext)
  - Accessible from Web components within a Web context
- Session
  - Accessible from Web components handling a request that belongs to the session
- Request
  - Accessible from Web components handling the request
- Page
  - Accessible from JSP page that creates the object

Web Context (ServletContext)

```
protected void doGet(HttpServletRequest request,
                     HttpServletResponse response) {

    ServletContext context = request.getServletContext();
    //. . .
}
```

- provides detail about execution environment of the servlets
- used to communicate with the container
  - e.g : reading a resource packaged in the web application, writing to a logfile, dispatching a request, ...

Example – getting a shared resource

```
public class CatalogServlet extends HttpServlet {
  private BookDB bookDB;
  public void init() throws ServletException {
    // Get context-wide attribute value from
    // ServletContext object
    bookDB = (BookDB)getServletContext().getAttribute("bookDB");
    if (bookDB == null) throw new
      UnavailableException("Couldn't get database.");
  }
}
```

Example - RequestDispatcher

```
public void doGet (HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {

    HttpSession session = request.getSession(true);
        ResourceBundle messages = (ResourceBundle)session.getAttribute("messages");

    // set headers and buffer size before accessing the Writer
    response.setContentType("text/html");
      response.setBufferSize(8192);
      PrintWriter out = response.getWriter();

    // then write the response
    out.println("<html>" +
          "<head><title>" + messages.getString("TitleBookDescription") +
          "</title></head>");

    // Get the dispatcher; it gets the banner to the user
    RequestDispatcher dispatcher =
        getServletContext().getRequestDispatcher("/banner");

    if (dispatcher != null)
        dispatcher.include(request, response);
    ...
```

Inclusion of a web ressource

Including Ressources

- Types of web ressources
  - Static resource

  - Dynamic web component (Servlet or JSP page)
    - Send the request to the “included” Web component
    - Execute the “included” Web component
    - Include the result of the execution from the “included” Web component in the response of the “including” servlet

Things that Included Web Resource can and cannot do

- Included Web resource has access to the request object, but it is limited in what it can do with the response
  - It can write to the body of the response and commit a response
  - It cannot set headers or call any method (for example, setCookie) that affects the headers of the response

Including another Web resource

- Get RequestDispatcher object from ServletContext object

  RequestDispatcher dispatcher =

   getServletContext().getRequestDispatcher("/banner");

- Then, invoke the include() method of the RequestDispatcher object passing request and response objects
  - dispatcher.include(request, response);

Example: BannerServlet as “Included” Web component

```
@WebServlet("/banner")
public class BannerServlet extends HttpServlet {
  public void doGet (HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {

    PrintWriter out = response.getWriter();
    out.println("<body bgcolor=\"#ffffff\">" +
    "<center>" + "<hr> <br> &nbsp;" + "<h1>" +
    "<font size=\"+3\" color=\"#CC0066\">Duke's </font>" +
    <img src=\"" + request.getContextPath() +
    "/duke.books.gif\">" +
    "<font size=\"+3\" color=\"black\">Bookstore</font>" +
    "</h1>" + "</center>" + "<br> &nbsp; <hr> <br> ");
  }
  public void doPost (HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {

    PrintWriter out = response.getWriter();
    out.println("<body bgcolor=\"#ffffff\">" +
    "<center>" + "<hr> <br> &nbsp;" + "<h1>" +
    "<font size=\"+3\" color=\"#CC0066\">Duke's </font>" +
    <img src=\"" + request.getContextPath() +
    "/duke.books.gif\">" +
    "<font size=\"+3\" color=\"black\">Bookstore</font>" +
    "</h1>" + "</center>" + "<br> &nbsp; <hr> <br> ");
  }
}
```

Example: Including “BannerServlet”

```
RequestDispatcher dispatcher =
  getServletContext().getRequestDispatcher("/banner");
if (dispatcher != null)
  dispatcher.include(request, response);
```

“Forwarding” to another Web resource

- To be used when one Web component do preliminary processing of a request and another component generate the response
- Should be used to give another resource responsibility for replying to the user
  - Throws an IllegalStateException if access to a ServletOutputStream or PrintWriter object have already been made within the servlet

“Forwarding” to another Web resource

- Get RequestDispatcher object from HttpServletRequest object
  - Set “request URL” to the path of the forwarded page

```
RequestDispatcher dispatcher
= request.getRequestDispatcher("/template.jsp");
```

- If the original URL is required for any processing, you can save it as a request attribute
- Invoke the forward() method of the RequestDispatcher object
  - `dispatcher.forward(request, response);`

Example: Dispatcher Servlet

```
public class Dispatcher extends HttpServlet {
  public void doGet(HttpServletRequest request,
    HttpServletResponse response) {
    request.setAttribute("selectedScreen",
      request.getServletPath());
    RequestDispatcher dispatcher = request.
      getRequestDispatcher("/template.jsp");
    if (dispatcher != null)
      dispatcher.forward(request, response);
  }
  public void doPost(HttpServletRequest request,
  …
}
```

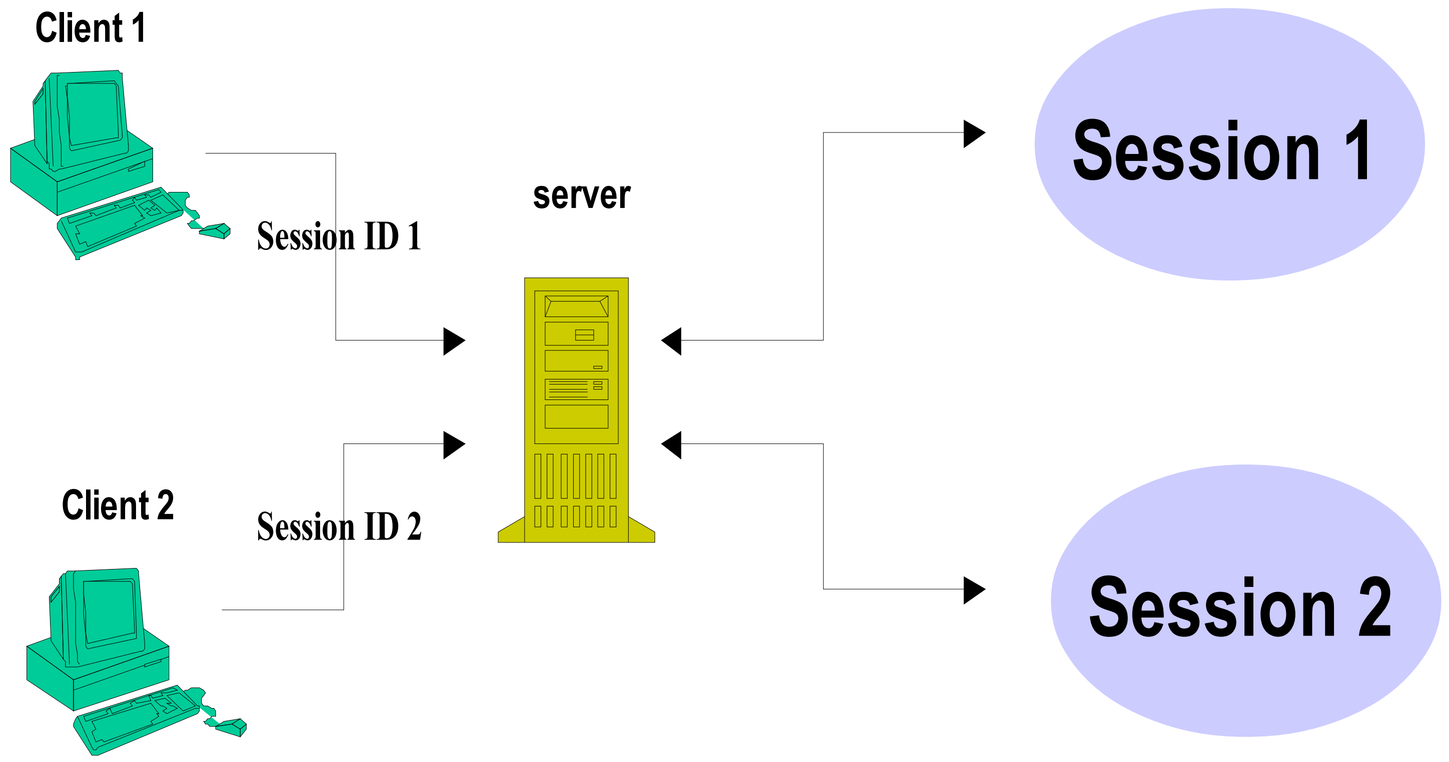
Part 5 : Session Tracking

- HTTP Session
- Setting and Getting Session Attributes
- Session Timeout
- Session Invalidation

Session Tracking

- HTTP is stateless protocol
  - Each time, a client talks to a web server, it opens a new connection
  - Server does not automatically maintains “conversational state” of a user

A Session Maintains Client Identity and State across multiple HTTP requests



HttpSession

- Maintains client state
  - Used by Servlets to set and get the values of session scope attributes
- To get a user's existing or new session object:
  - HttpSession session = request.getSession(true);
    - "true" means the server should create a new session object if necessary

Example: Getting HttpSession Object

```
public class CatalogServlet extends HttpServlet {
   public void doGet (HttpServletRequest request,
                  HttpServletResponse response)
     throws ServletException, IOException {

     // Get the user's session and shopping cart
     HttpSession session =request.getSession(true);
           ...
     out = response.getWriter();
           ...
       }
   }
   ...
```

HttpSession Java Interface

- Contains Methods to
  - View and manipulate information about a session, such as the session identifier, creation time, and last accessed time
  - Bind objects to sessions, allowing user information to persist across multiple user connections
- To stores values:
  - `session.setAttribute("cartItem", cart);`
- To retrieves values:
  - `session.getAttribute("cartItem");`

Setting and Getting Attribute

```
public class CatalogServlet extends HttpServlet {
   public void doGet (HttpServletRequest request,
                             HttpServletResponse response)
                         throws ServletException, IOException {
     // Get the user's session and shopping cart
     HttpSession session = request.getSession(true);
     ShoppingCart cart = (ShoppingCart)session.getAttribute(
                            "examples.bookstore.cart");
     // If the user has no cart, create a new one
     if (cart == null) {
         cart = new ShoppingCart();
         session.setAttribute("examples.bookstore.cart", cart);
     }
     ...
     //see next slide.
   }
}
```

Session Timeout

- Used when an end-user can leave the browser without actively closing a session
- Sessions usually times out after 30 minutes of inactivity
  - Product specific
  - A different timeout may be set by server admin
- getMaxInactiveInterval(), setMaxInactiveInterval() methods of HttpSession interface
  - Gets or sets the amount of time, session should go without access before being invalidated

Session Invalidation

- public void invalidate()
  - Expire the session and unbinds all objects with it
- Caution
  - Remember that a session object is shared by multiple servlets/JSP-pages and invalidating it could destroy data that other servlet/JSP-pages are using

Example: Invalidate a Session

```
 public class ReceiptServlet extends HttpServlet {
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
       throws ServletException, IOException {
                ...
       scart = (ShoppingCart)
          session.getAttribute("examples.bookstore.cart");
                ...
       // Clear out shopping cart by invalidating the session
       session.invalidate();

       // set content type header before accessing the Writer
       response.setContentType("text/html");
       out = response.getWriter();
       ...
    }
}
```

Part 6 : Filters, Listeners, Asynchronous Support

- Servlet Filters
- Listeners
- Asynchronous Support

Servlet Filters

```
@WebFilter("/*")
public class LoggingFilter implements javax.servlet.Filter {
    public void doFilter(HttpServletRequest request,
                         HttpServletResponse response) {
        //. . .
    }
}
```

- update the request and response payload and header information from and to the servlet
  - e.g : for logging, data compression, and encryption, ...

Servlet Filters

```
<filter>
    <filter-name>LoggingFilter</filter-name>
    <filter-class>org.sample.LoggingFilter</filter-class>
</filter>
. . .
<filter-mapping>
    <filter-name>LoggingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Servlet Filters

```
public class MyInitializer implements ServletContainerInitializer {
    public void onStartup (Set<Class<?>> clazz,
                                ServletContext context) {
        FilterRegistration.Dynamic reg =
            context.addFilter("LoggingFilter",
                    "org.example.LoggingFilter");
        reg.addMappingForUrlPatterns(null, false, "/");
    }
}
```

Event Listeners

- Provide life-cycle callback events for *ServletContext* , *HttpSession, ServletRequest* objects
  - supports event notifications for state changes
- Specified with :
  - *@WebListener* annotation,
  - Declaration in *web.xml*, or
  - registration via *ServletContext.addListener* methods

Listener Interfaces

- ServletContextListener
  - contextInitialized/Destroyed(ServletContextEvent)
- ServletContextAttributeListener
  - attributeAdded/Removed/Replaced(
    ServletContextAttributeEvent)
- HttpSessionListener
  - sessionCreated/Destroyed(HttpSessionEvent)
- HttpSessionAttributeListener
  - attributedAdded/Removed/Replaced(
    HttpSessionBindingEvent)
- HttpSessionActivationListener
  - Handles sessions migrate from one server to another
  - sessionWillPassivate(HttpSessionEvent)
  - sessionDidActivate(HttpSessionEvent)

Event Listeners

```
@WebListener
public class MyContextListener implements ServletContextListener {
    @Override
    public void contextInitialized(ServletContextEvent sce) {
        ServletContext context = sce.getServletContext();
        //. . .
    }
    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        //. . .
    }
}
```

Event Listeners

```
@WebListener
public class MyServletContextAttributeListener
                    implements ServletContextAttributeListener {
     @Override
     public void attributeAdded(ServletContextAttributeEvent event) {
          //. . . event.getName();
          //. . . event.getValue();
     }
     @Override
     public void attributeRemoved(ServletContextAttributeEvent
                                                          event) {
          //. . .
     }
     @Override
     public void attributeReplaced(ServletContextAttributeEvent
                                                          event) {
          //. . .
     }
}
```

Asynchronous Support

```
@WebServlet(urlPatterns="/async", asyncSupported=true)
public class MyAsyncServlet extends HttpServlet {
	//. . .
}
```

- To handle long-running process without wasting valuable server ressources
  - running thread waiting for completion

Asynchronous Support

```
class MyAsyncService implements Runnable {
    AsyncContext ac;
    public MyAsyncService(AsyncContext ac) {
        this.ac = ac;
    }
    @Override
    public void run() {
        //. . .
        ac.complete();
    }
}
```

Asynchronous Support

```
@Override
protected void doGet(HttpServletRequest request,
                              HttpServletResponse response) {
    AsyncContext ac = request.startAsync();
    ac.addListener(new AsyncListener() {
        public void onComplete(AsyncEvent event)
            throws IOException {
            //. . .
        }
        public void onTimeout(AsyncEvent event)
            throws IOException {
            //. . .
        }
        //. . .
    });
    ScheduledThreadPoolExecutor executor = new
        ScheduledThreadPoolExecutor(10);
    executor.execute(new MyAsyncService(ac));
}
```

More Information on Servlets

- Servlets Specification JSR340
  - https://jcp.org/aboutJava/communityprocess/final/jsr340/index.html