Servlet Basics I

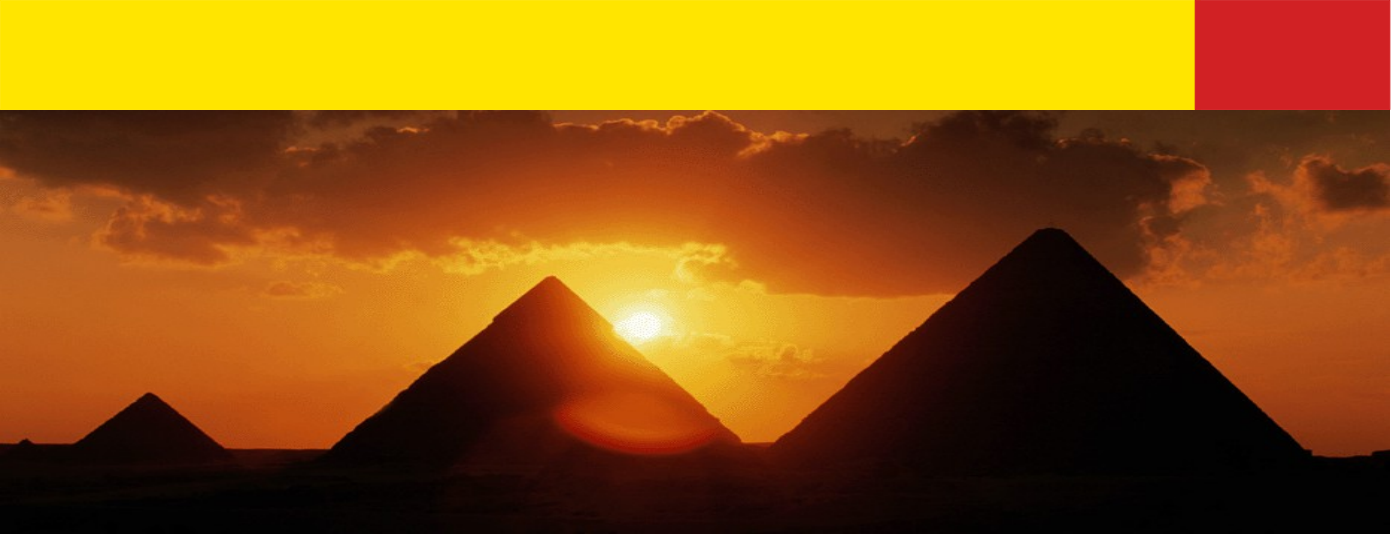
Sang Shin
JPassion.com
"Code with Passion!"

Topics

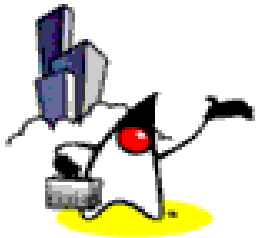
- Covered in Servlet Basics I
  - What is Servlet?
  - Servlet in big picture within a Java EE application
  - Servlet request & response model
  - Servlet life cycle
  - Servlet request
  - Servlet request URL
- Covered in Servlet Basics II
  - Servlet response: Status, Header, Body
  - Servlet scope objects
  - Error Handling

Advanced Topics:

- Session Tracking
- Servlet Filters
- Servlet life-cycle events
- Including, forwarding to, and redirecting to other web resources
- Concurrency Issues
- Invoker Servlet



What is Servlet?



What is Servlet?

- Java™ objects which are based on Servlet framework and APIs
- Extend the functionality of a HTTP server for creating dynamic contents
- Mapped to URLs and managed by container with a simple architecture
- Available and running on all major web servers and app servers
- Platform and server independent

Static vs. Dynamic Contents

- Static contents
  - Typically static HTML page
  - Same display for everyone
- Dynamic contents
  - Contents is dynamically generated based on conditions
  - Conditions could be
    - User identity
    - Time of the day
    - User entered values through forms and selections
  - Examples
    - ETrade webpage customized just for you, my Yahoo

First Servlet Code

```
Public class HelloServlet extends HttpServlet {

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response){

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<title>Hello World!</title>");

  }
  ...
}
```

CGI versus Servlet

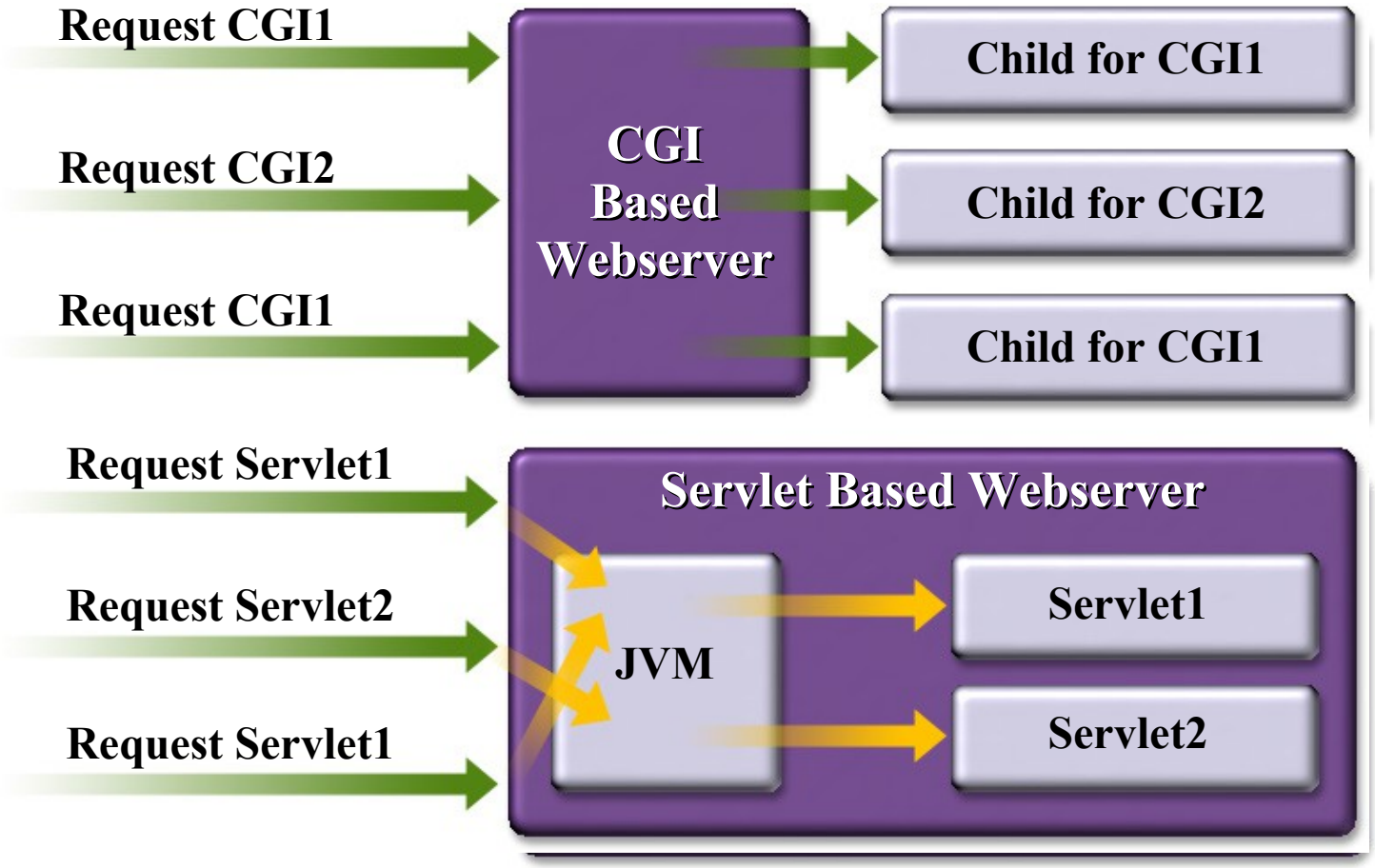
CGI

- Written in C, C++, Visual Basic and Perl
- Difficult to maintain, non-scalable, non-manageable
- Prone to security problems of programming language
- Resource intensive and inefficient
- Platform and application-specific

Servlet

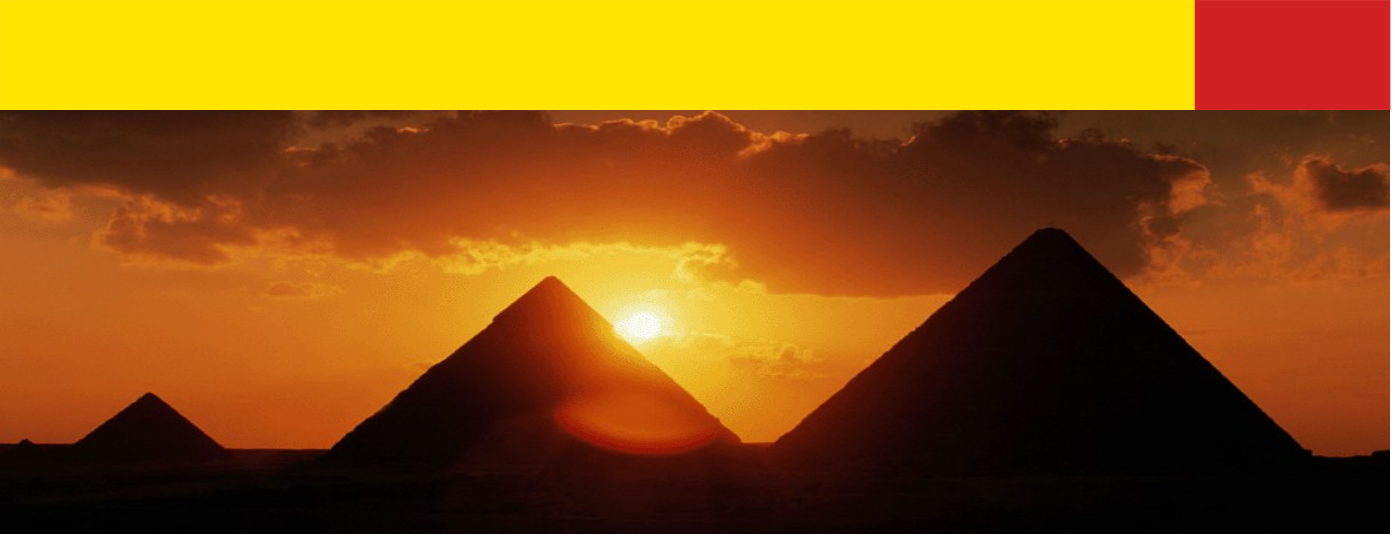
- Written in Java
- Powerful, reliable, and efficient
- Improves scalability, reusability (component based)
- Leverages built-in security of Java programming language
- Platform independent and portable

Servlet vs. CGI

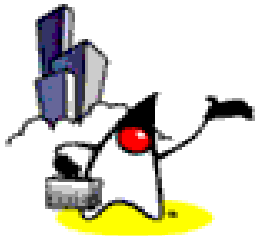


Advantages of Servlet over CGI

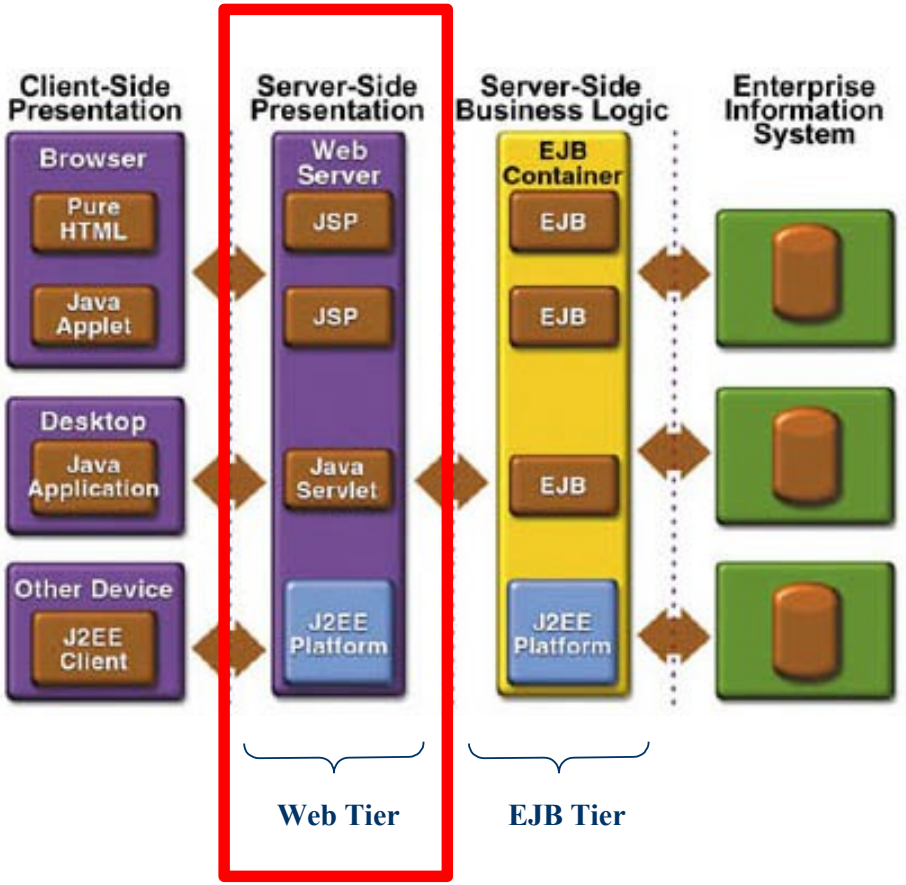
- No CGI limitations
- Abundant third-party tools and Web servers supporting Servlet
- Access to entire family of Java APIs
- Reliable, better performance and scalability
- Platform and server independent
- Secure
- Most servers allow automatic reloading of Servlet's by administrative action

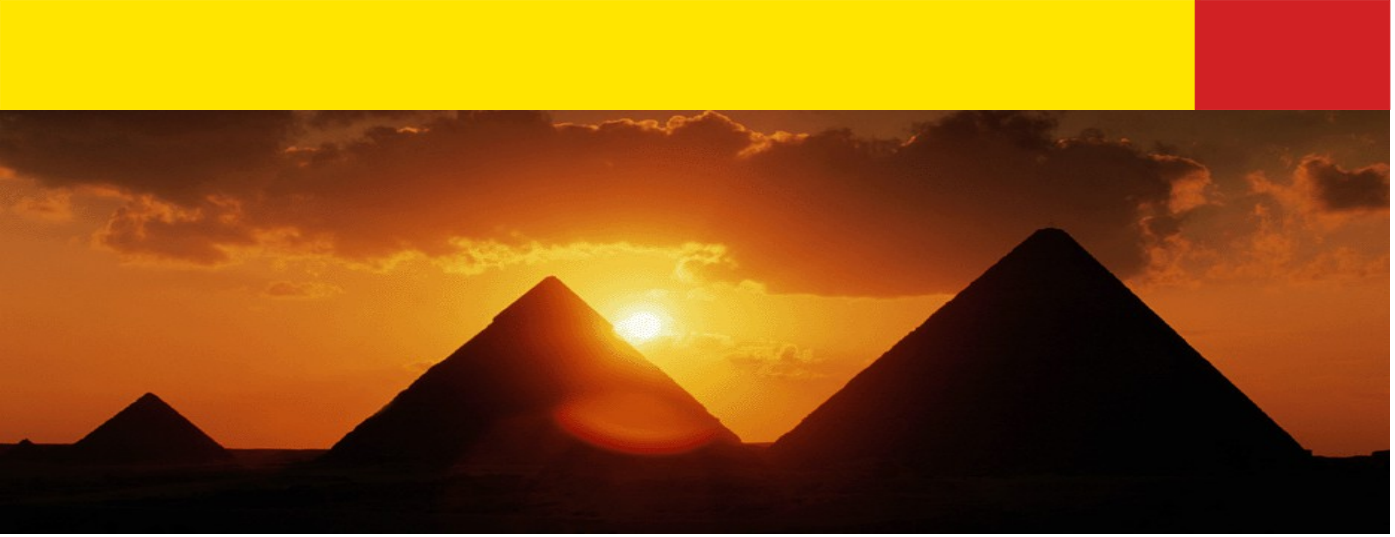


Servlet in a Big Picture of Java EE Application

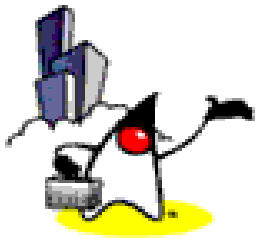


Where are Servlet and JSP?

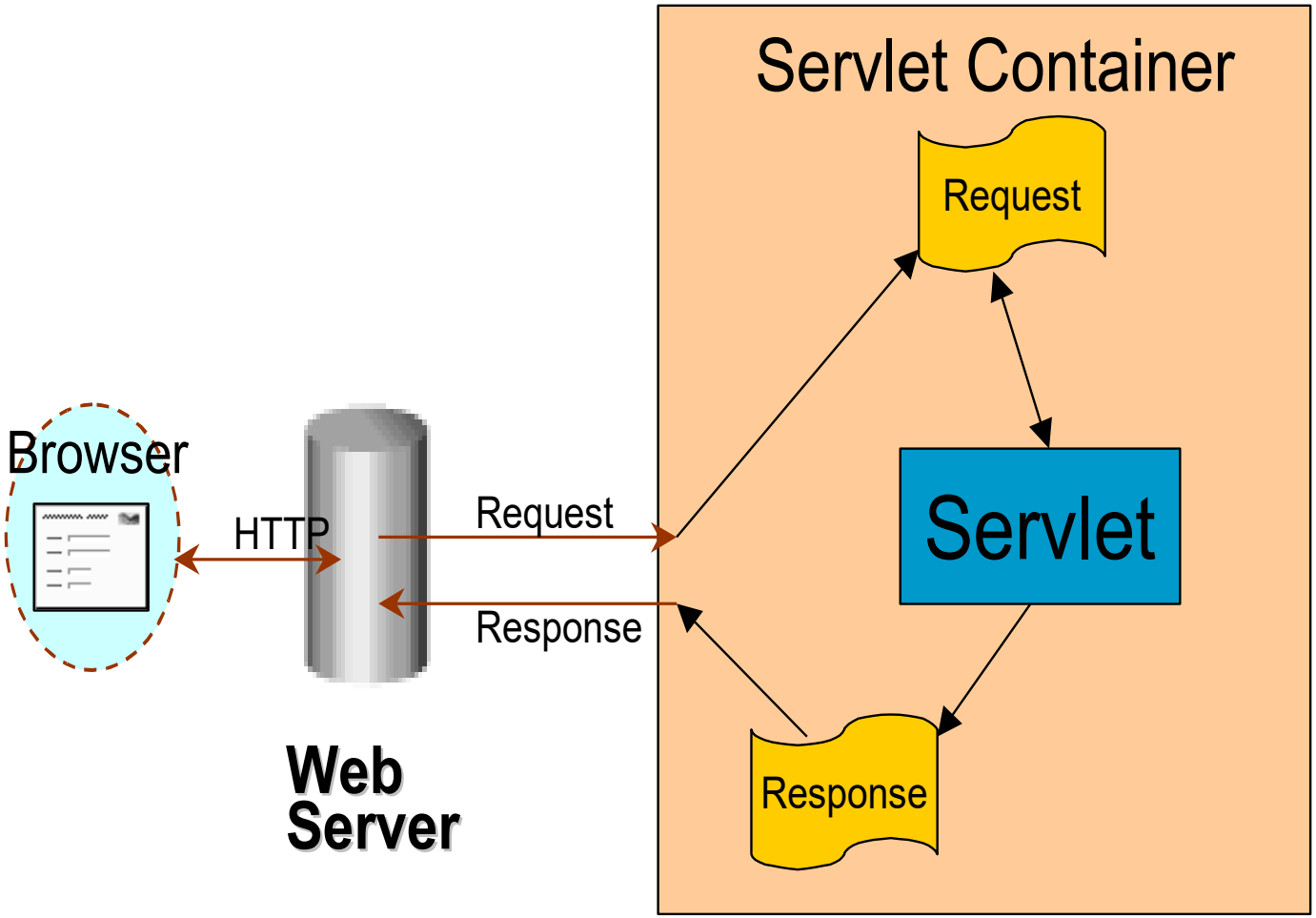




Servlet Request & Response Model



Servlet Request and Response Model



What does Servlet Do?

1. Receives client request (mostly in the form of HTTP request)
2. Extract some information from the request
3. Do perform business logic processing (possibly by accessing database, invoking EJBs, web services etc)
4. Create and send response to client (mostly in the form of HTTP response) or forward the request to another servlet or JSP page

Requests and Responses

- What is a request?
  - Information that is sent from client to a server
    - Who made the request
    - What user-entered data is sent
    - Which HTTP headers are sent
- What is a response?
  - Information that is sent to client from a server
    - Text(html, plain) or binary(image) data
    - HTTP headers, cookies, etc

HTTP

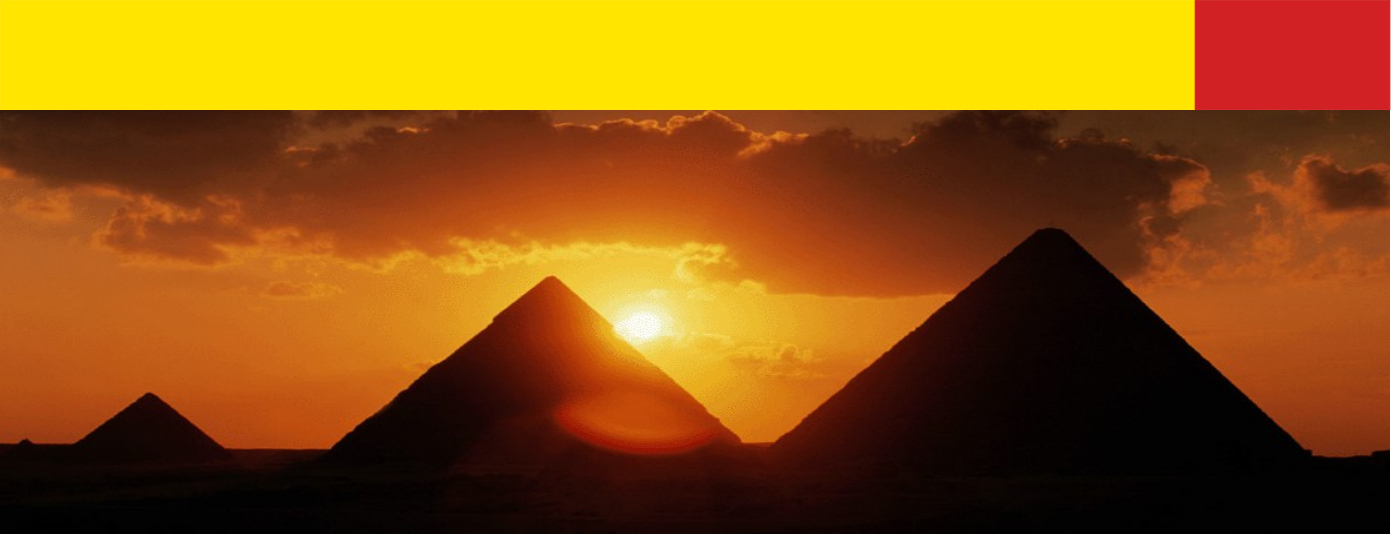
- HTTP request contains
  - header
  - a HTTP method
    - Get: Input form data is passed as part of URL
    - Post: Input form data is passed within message body
    - Put
  - request data

HTTP GET and POST

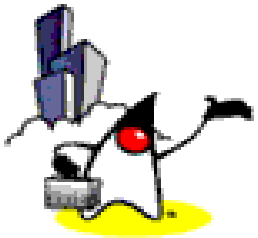
- The most common HTTP methods
- HTTP GET requests:
  - User entered information is appended to the URL in a query string
  - Can only send limited amount of data as query param's
    - .../servlet/ViewCourse?FirstName=Sang&LastName=Shin
- HTTP POST requests:
  - User entered information is sent as data (not appended to URL)
  - Can send any amount of data

First Servlet Again

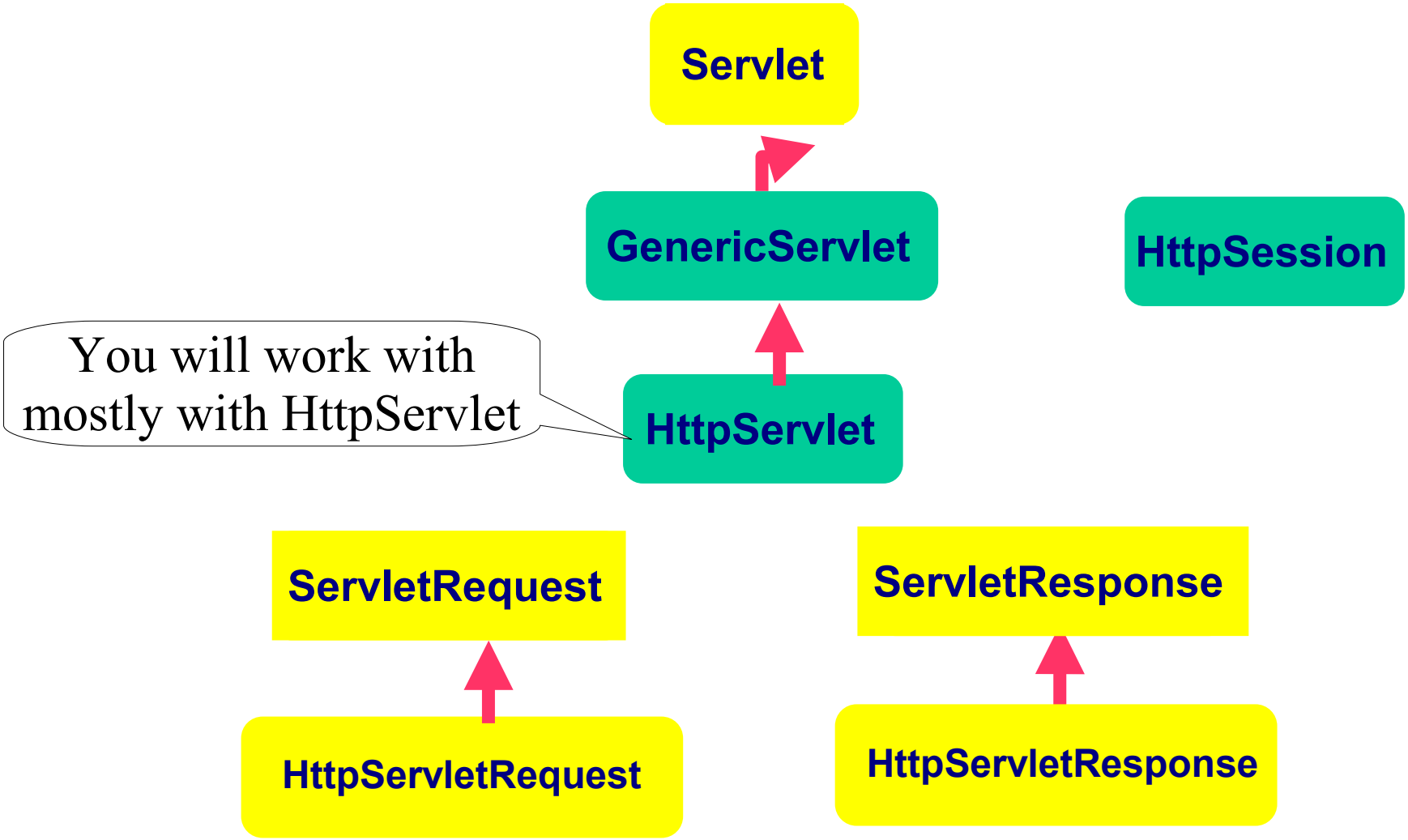
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

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>");

  }
}
```

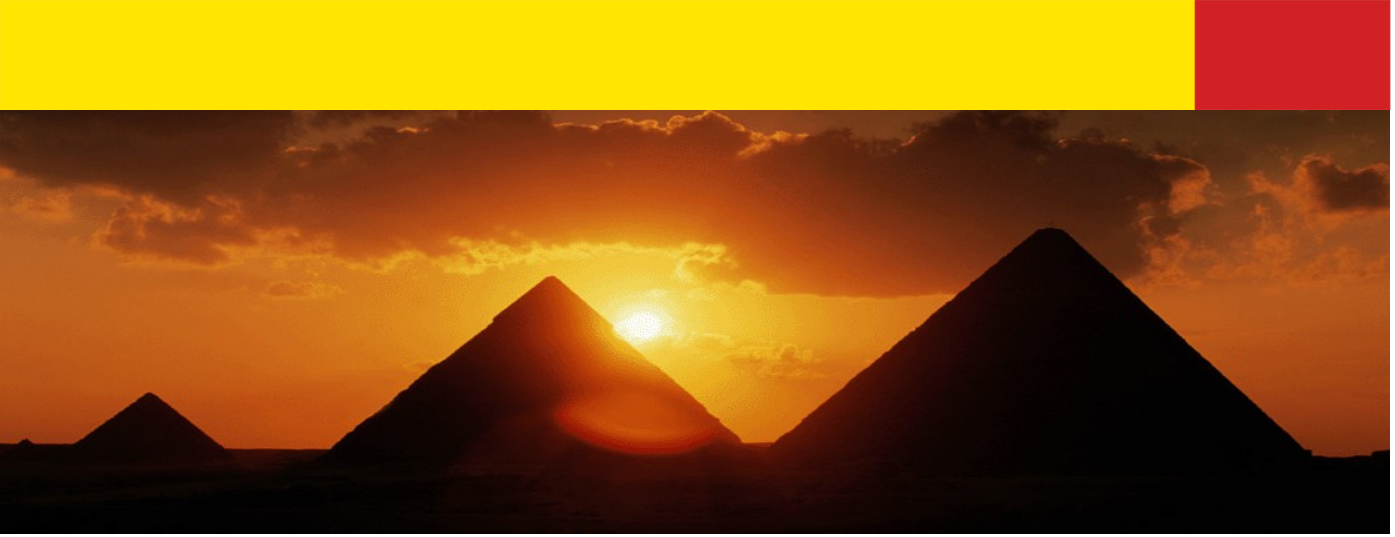


Interfaces & Classes of Servlet

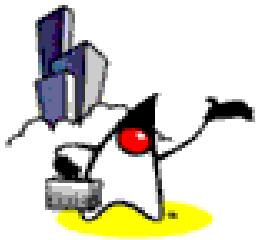


Servlet Interfaces & Classes

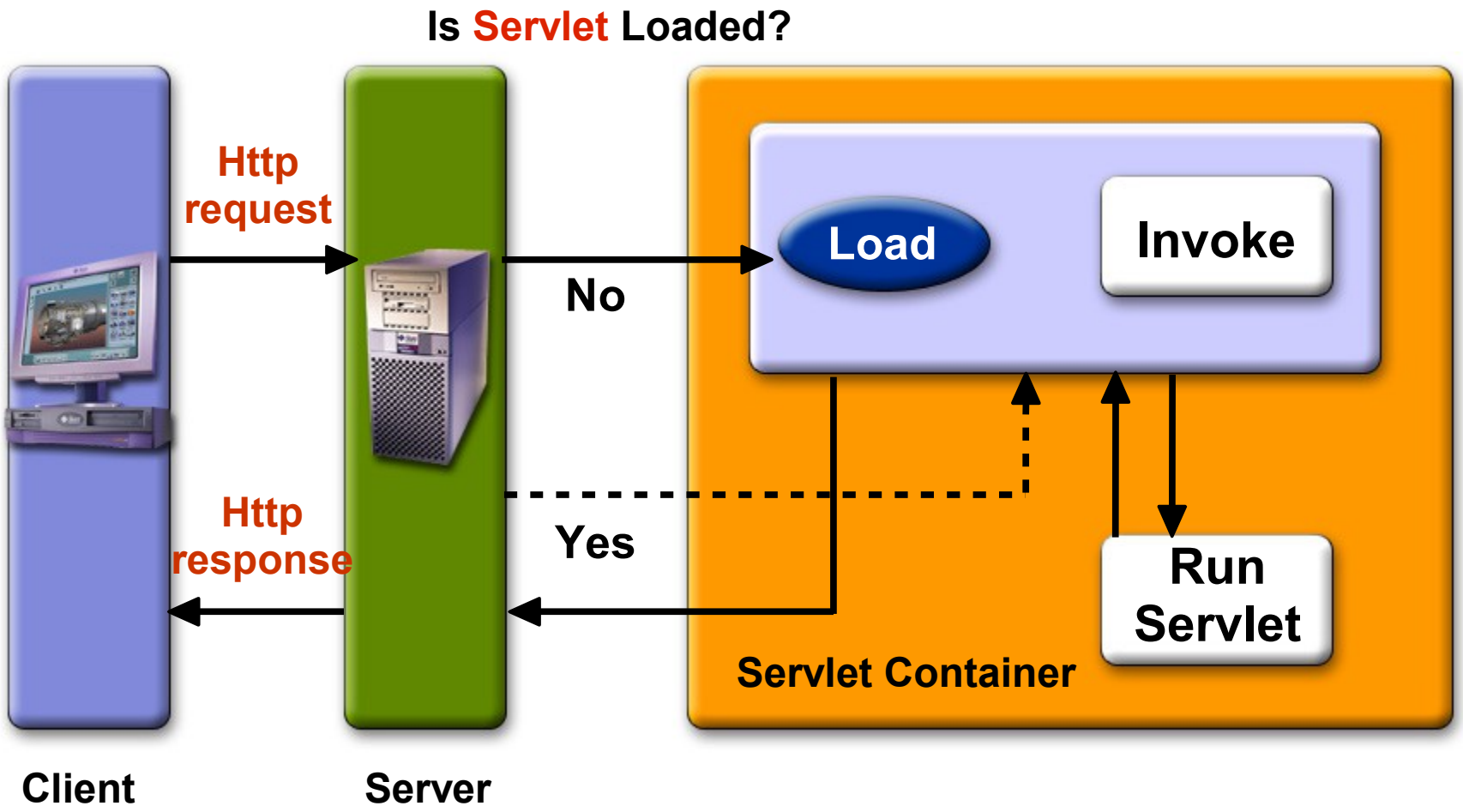




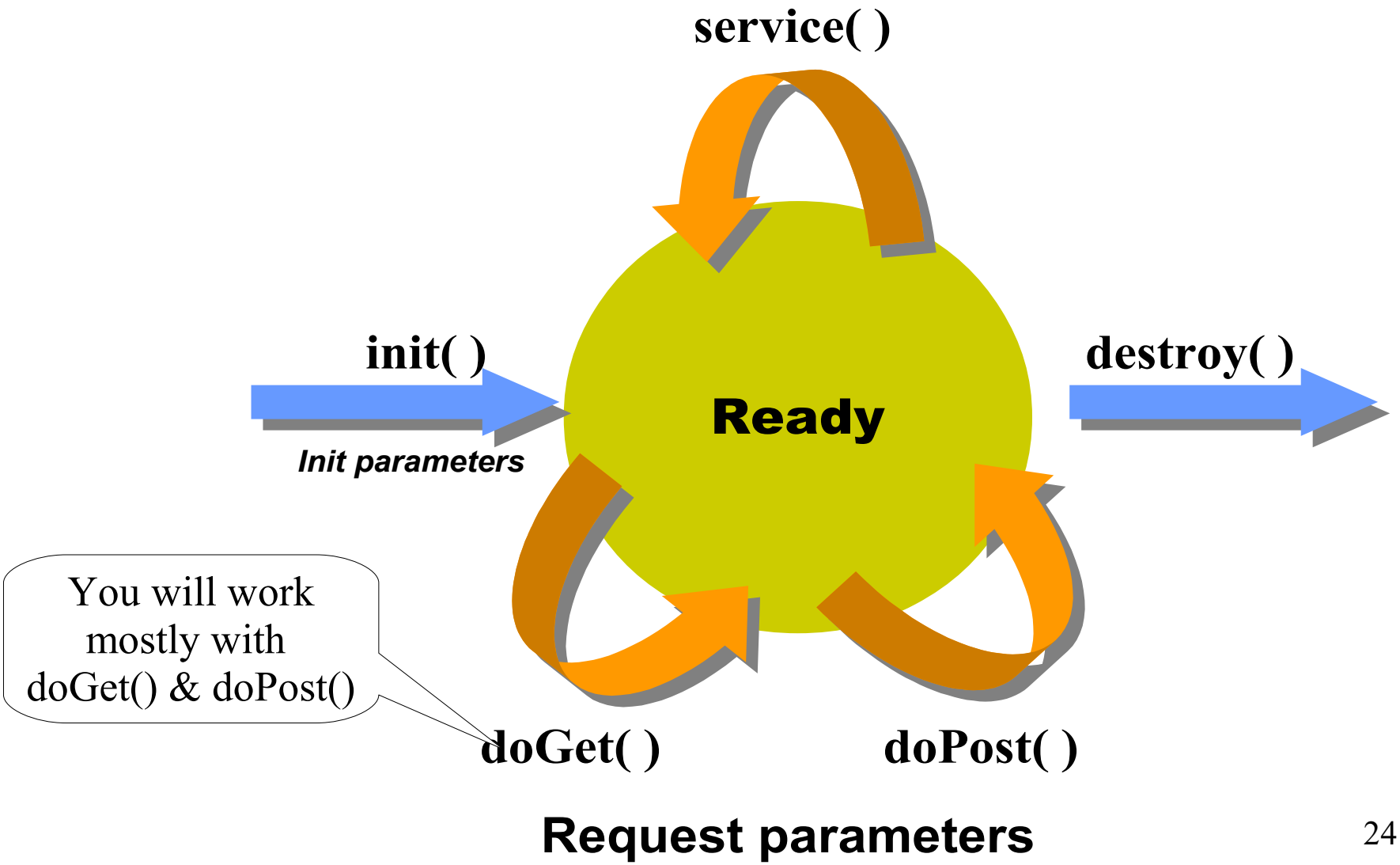
Servlet Life-Cycle



Servlet Life-Cycle



HttpServlet Life Cycle Methods



Servlet Life Cycle Methods

- Invoked by container
  - Container controls life cycle of a servlet
- Defined in
  - javax.servlet.GenericServlet class or
    - init()
    - destroy()
    - service() - this is an abstract method
  - javax.servlet.http.HttpServlet class
    - service() - implementation
    - doGet(), doPost(), doXxx()

Servlet Life Cycle Methods

- init()
  - Invoked once when the servlet is first instantiated
  - Perform any set-up in this method
    - Read init parameters
    - Setting up a database connection
- destroy()
  - Invoked before servlet instance is removed
  - Perform any clean-up
    - Closing a previously created database connection

Example: init() reading Configuration parameters

```
public void init(ServletConfig config) throws
   ServletException {
        super.init(config);
        String driver = getInitParameter("driver");
        String fURL = getInitParameter("url");
        try {
          openDBConnection(driver, fURL);
        } catch (SQLException e) {
           e.printStackTrace();
        } catch (ClassNotFoundException e){
           e.printStackTrace();
        }
 }
```

Setting Init Parameters in web.xml

```
<web-app>
    <servlet>
        <servlet-name>chart</servlet-name>
        <servlet-class>ChartServlet</servlet-class>
        <init-param>
            <param-name>driver</param-name>
            <param-value>
              COM.cloudscape.core.RmiJdbcDriver
            </param-value>
        </init-param>
        <init-param>
            <param-name>url</param-name>
            <param-value>
              jdbc:cloudscape:rmi:CloudscapeDB
            </param-value>
        </init-param>
    </servlet>
    ...
</web-app>
```

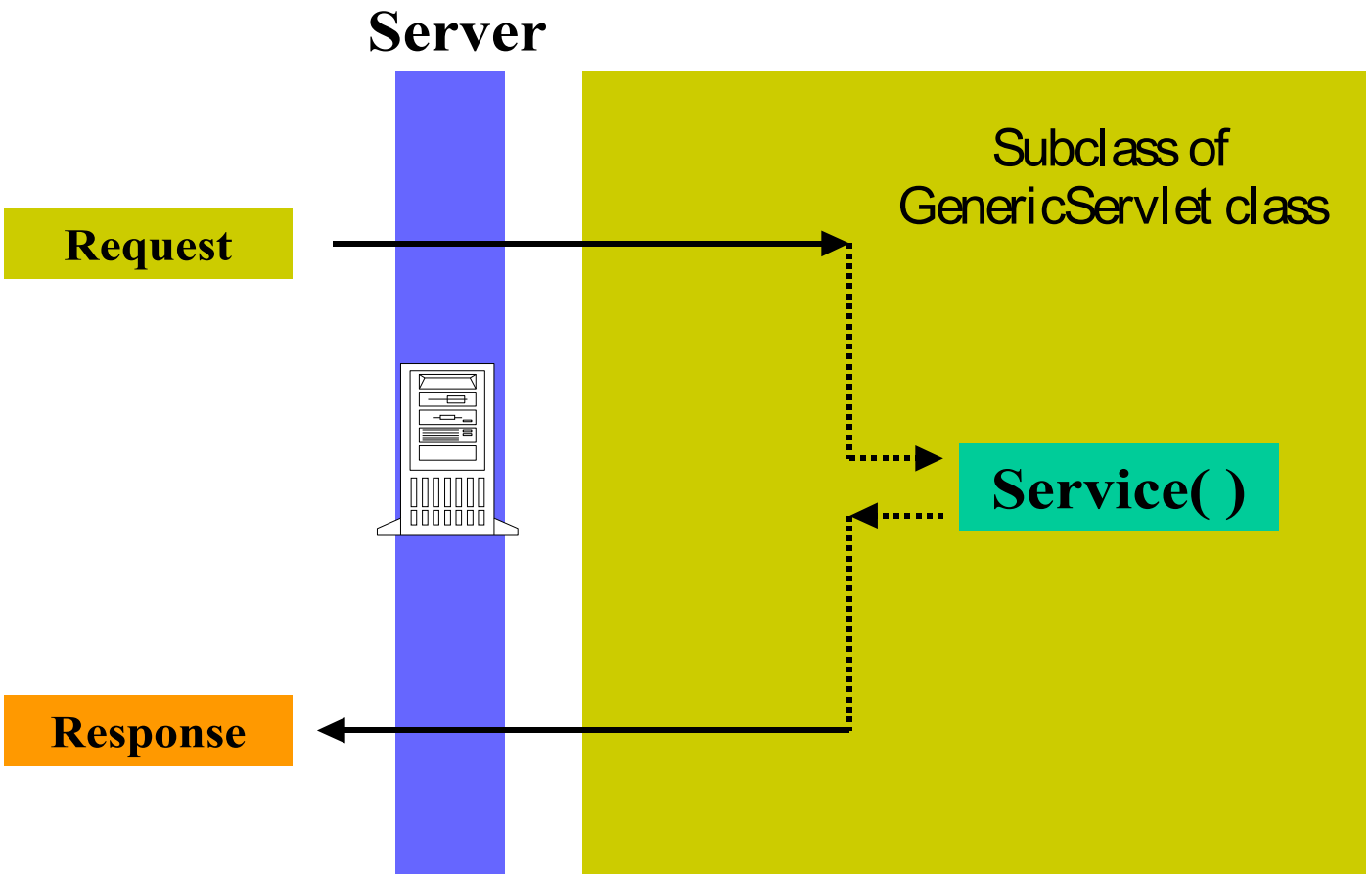
Servlet Life Cycle Methods

- service() javax.servlet.GenericServlet class
  - Abstract method
- service() in javax.servlet.http.HttpServlet class
  - Concrete method (implementation)
  - Dispatches to doGet(), doPost(), etc
  - Do not override this *service()* method!
- doGet(), doPost(), doXxx() in javax.servlet.http.HttpServlet
  - Handles HTTP GET, POST, etc. requests
  - Override these methods in your servlet to provide desired behavior

service() & doGet()/doPost()

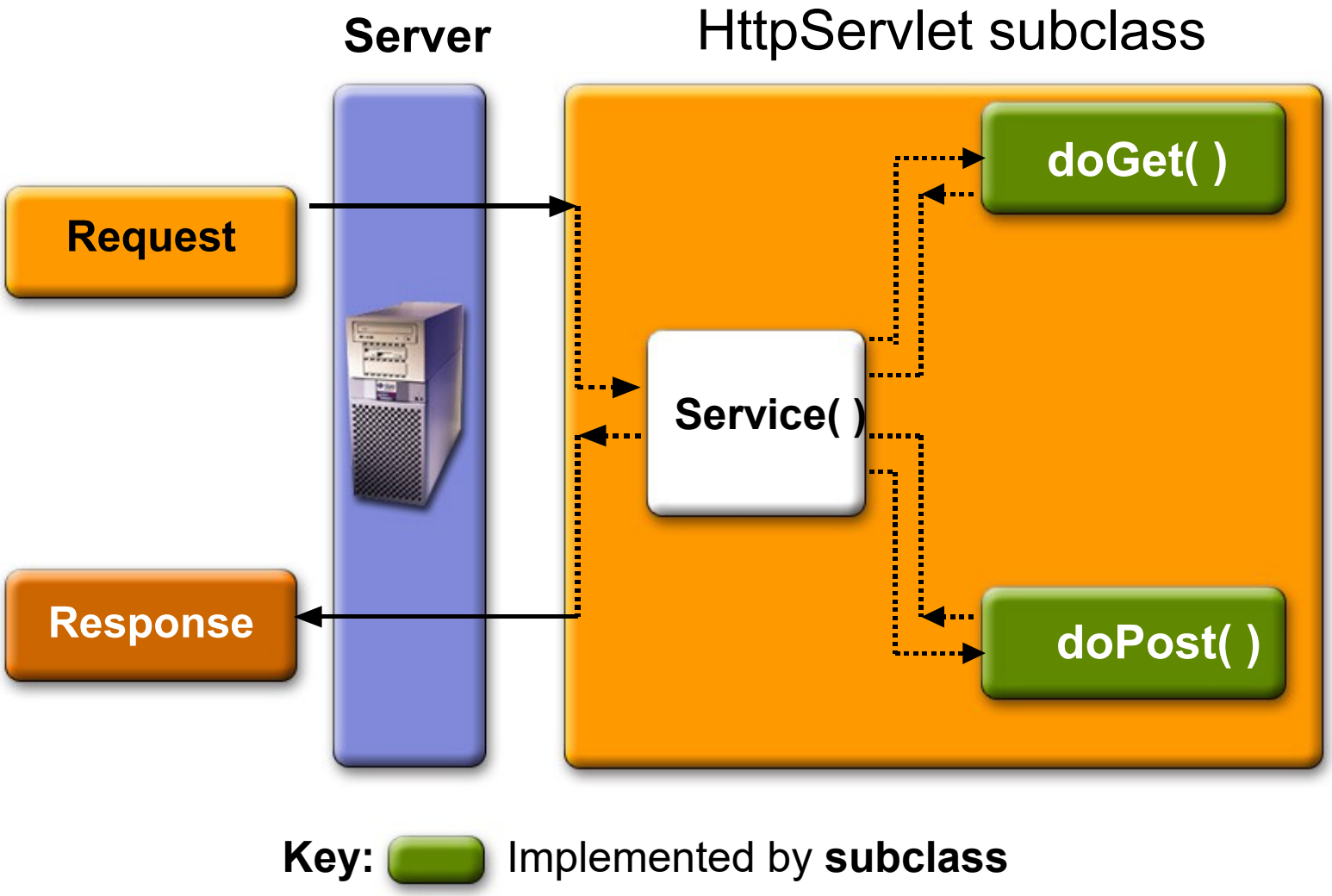
- service() methods take generic requests and responses:
  - service(ServletRequest request, ServletResponse response)
- doGet() or doPost() take HTTP requests and responses:
  - doGet(HttpServletRequest request, HttpServletResponse response)
  - doPost(HttpServletRequest request, HttpServletResponse response)

Service() Method



Key: Implemented by **subclass**

doGet() and doPost() Methods



Typical Things You Do in doGet() & doPost()

- Extract client-sent information (HTTP parameter) from HTTP request
- Set (Save) and get (read) attributes to/from Scope objects
- Perform some business logic or access database
- Optionally forward the request to other Web components (Servlet or JSP)
- Populate HTTP response message and send it to client

Example: Simple doGet()

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

Public class HelloServlet extends HttpServlet {

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

    // Just send back a simple HTTP response
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<title>First Servlet</title>");
    out.println("<big>Hello J2EE Programmers! </big>");
  }
}
```

Example: Sophisticated doGet()

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

        // Read session-scope attribute "messages"
        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 (Populate the header part of 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);
```

Example: Sophisticated doGet()

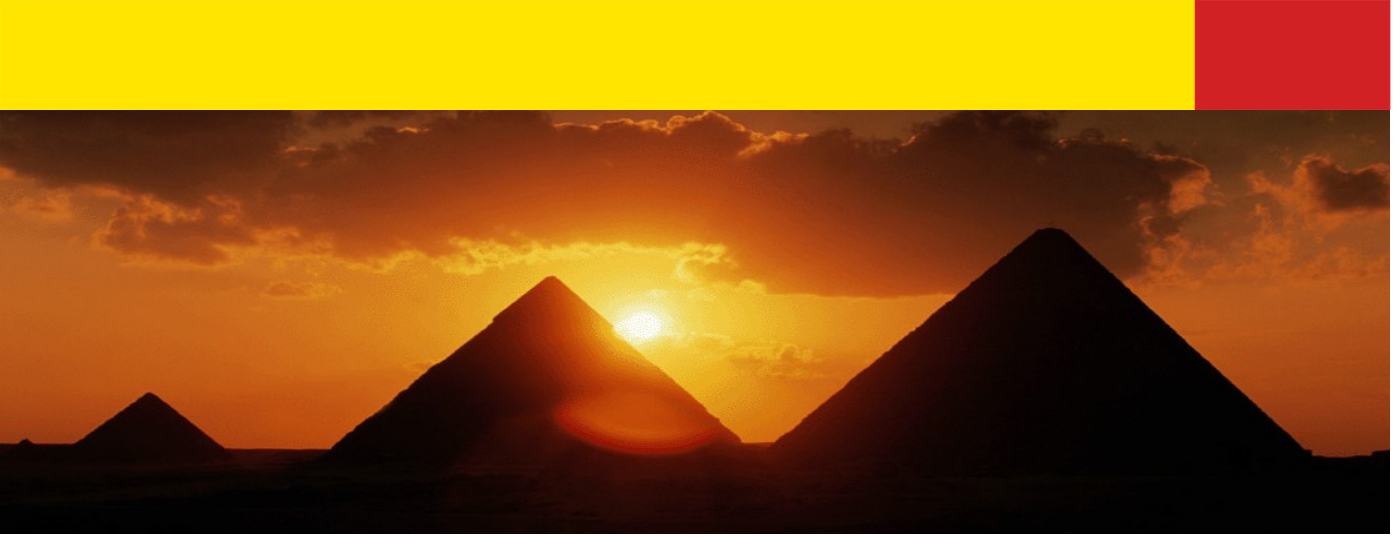
```
        // Get request parameter (Get the identifier of the book to display)
        String bookId = request.getParameter("bookId");
        if (bookId != null) {

            // and the information about the book (Perform business logic)
            try {
               BookDetails bd = bookDB.getBookDetails(bookId);
               Currency c = (Currency)session.getAttribute("currency");
               if (c == null) {
                  c = new Currency();
                  c.setLocale(request.getLocale());
                  session.setAttribute("currency", c);
               }
               c.setAmount(bd.getPrice());

               // Print out the information obtained
               out.println("...");
            } catch (BookNotFoundException ex) {
                     response.resetBuffer();
                     throw new ServletException(ex);
            }

        }
        out.println("</body></html>");
        out.close();
    }
```

Steps of Populating HTTP Response

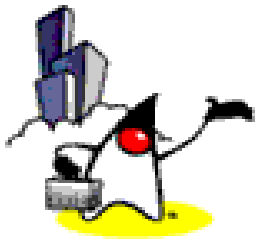
- Fill Response headers
- Set some properties of the response
  - Buffer size
- Get an output stream object from the response
- Write body content to the output stream

Example: Simple Response

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

    // Fill response headers
    response.setContentType("text/html");
    // Set buffer size
    response.setBufferSize(8192);
    // Get an output stream object from the response
    PrintWriter out = response.getWriter();
    // Write body content to output stream
    out.println("<title>First Servlet</title>");
    out.println("<big>Hello J2EE Programmers! </big>");
  }
}
```



Servlet Request (HttpServletRequest)

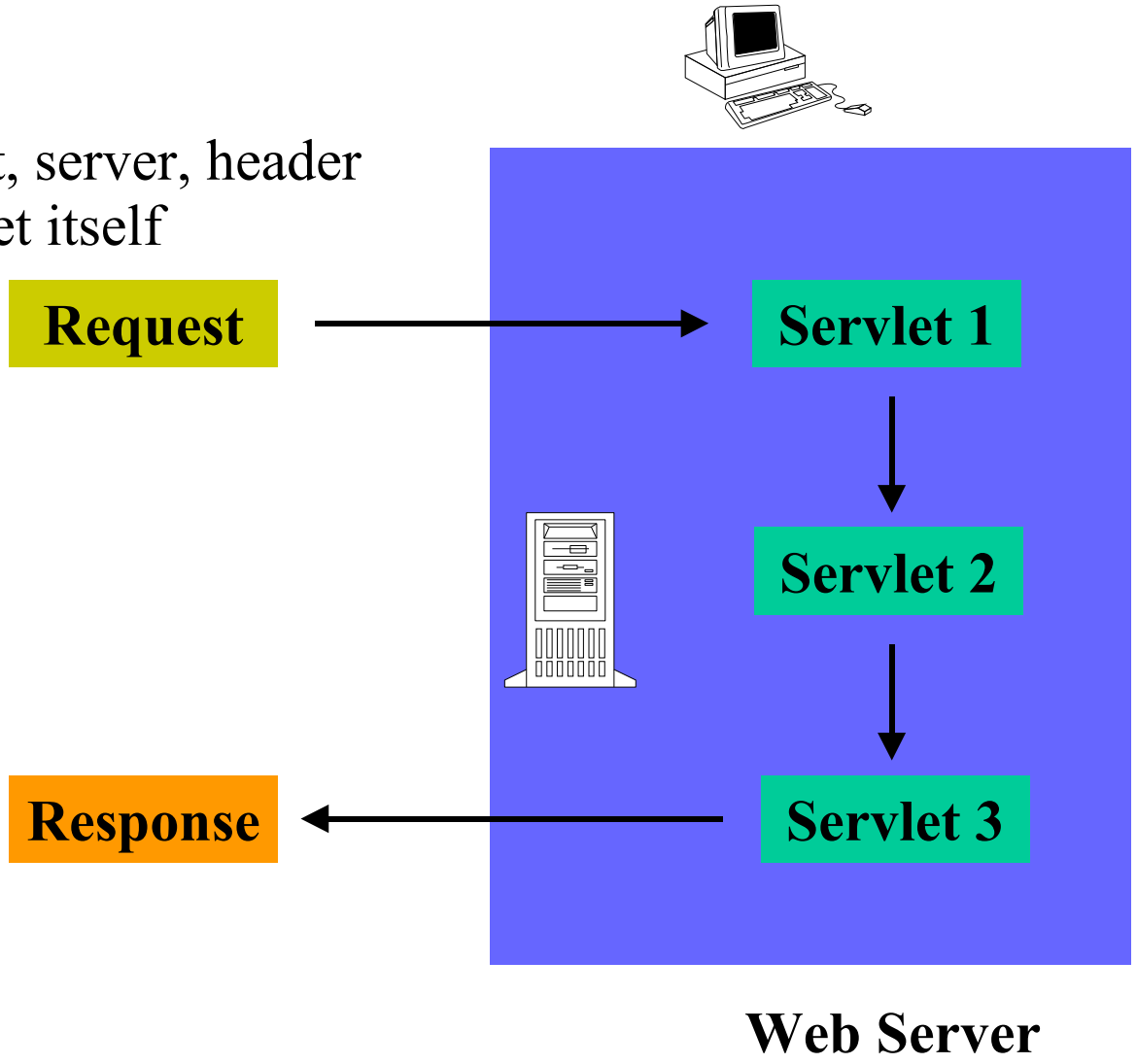


What is Servlet Request?

- Contains data passed from client to servlet
- All servlet requests implement ServletRequest interface which defines methods for accessing
  - Client sent parameters
  - Object-valued attributes
  - Locales
  - Client and server
  - Input stream
  - Protocol information
  - Content type
  - If request is made over secure channel (HTTPS) or not

Requests

data,
client, server, header
servlet itself



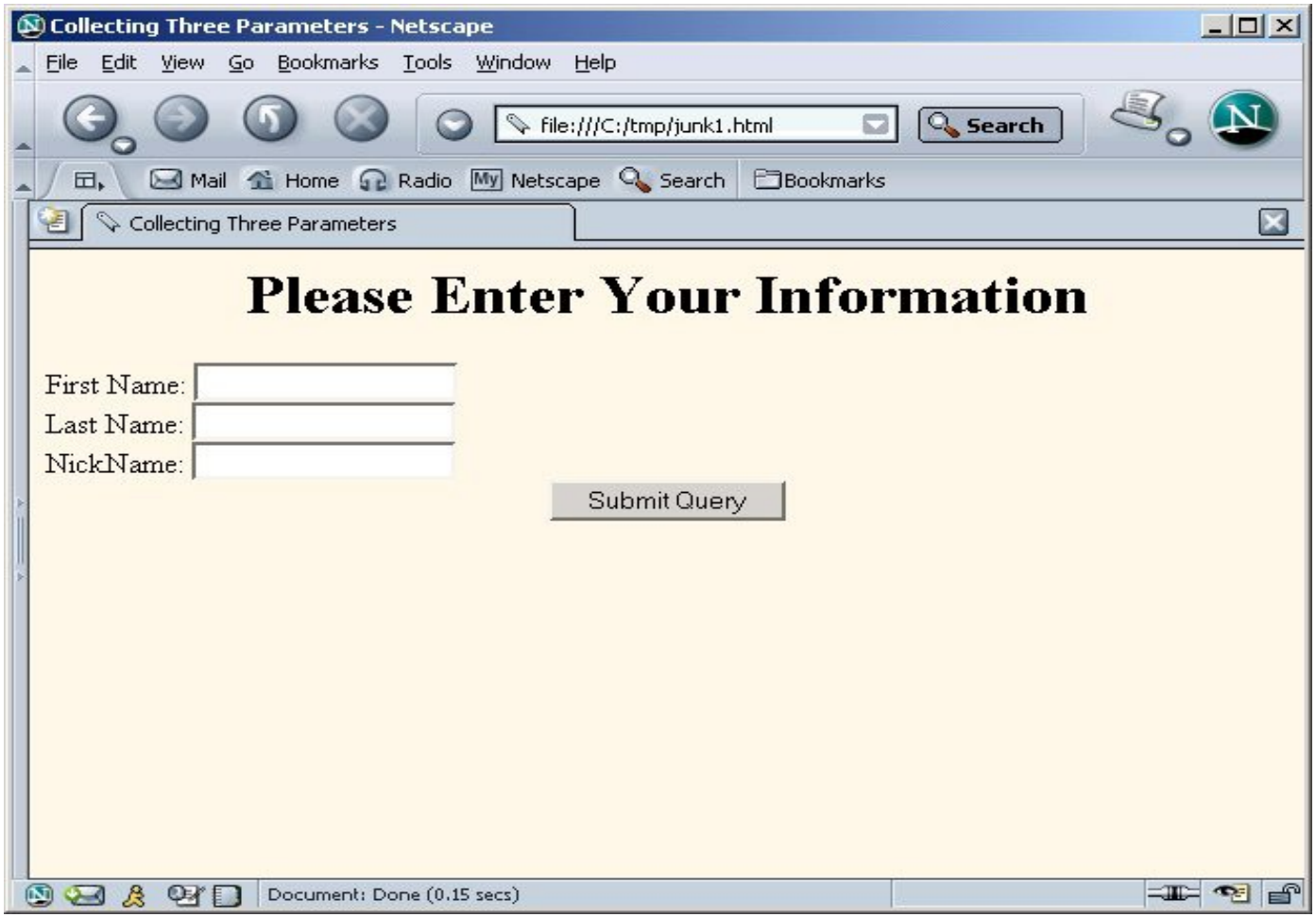
Getting Client Sent Parameters

- A request can come with any number of parameters
- Parameters are sent from HTML forms:
  - GET: as a query string, appended to a URL
  - POST: as encoded POST data, not appeared in the URL
- *getParameter("parameterName")*
  - Returns the value of parameterName
  - Returns null if no such parameter is present
  - Works identically for GET and POST requests

A Sample FORM using GET

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
 <HTML>
 <HEAD>
   <TITLE>Collecting Three Parameters</TITLE>
 </HEAD>
 <BODY BGCOLOR="#FDF5E6">
 <H1 ALIGN="CENTER">Please Enter Your Information</H1>

 <FORM ACTION="/sample/servlet/ThreeParams">
   First Name:  <INPUT TYPE="TEXT" NAME="param1"><BR>
   Last Name:   <INPUT TYPE="TEXT" NAME="param2"><BR>
   Class Name:  <INPUT TYPE="TEXT" NAME="param3"><BR>
   <CENTER>
     <INPUT TYPE="SUBMIT">
   </CENTER>
 </FORM>

 </BODY>
 </HTML>
```

A Sample FORM using GET



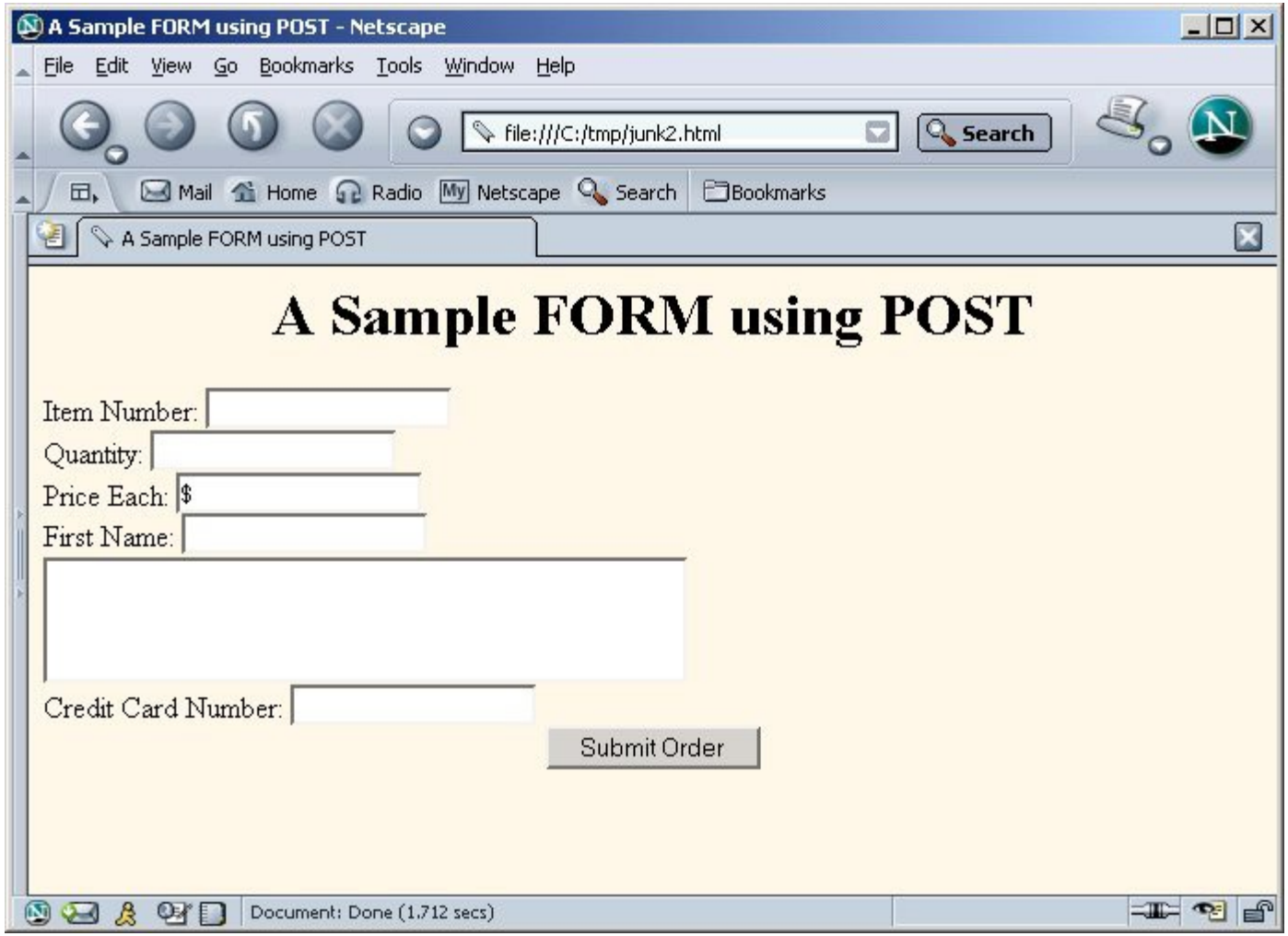
A FORM Based Servlet: Get

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
/** Simple servlet that reads three parameters from the html form */
public class ThreeParams extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
                    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Your Information";
    out.println("<HTML>" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
                "<UL>\n" +
                "  <LI><B>First Name in Response</B>: "
                + request.getParameter("param1") + "\n" +
                "  <LI><B>Last Name in Response</B>: "
                + request.getParameter("param2") + "\n" +
                "  <LI><B>NickName in Response</B>: "
                + request.getParameter("param3") + "\n" +
                "</UL>\n" +
                "</BODY></HTML>");
  }
}
```

A Sample FORM using POST

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>A Sample FORM using POST</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">A Sample FORM using POST</H1>
<FORM ACTION="/sample/servlet/ShowParameters" METHOD="POST">
  Item Number: <INPUT TYPE="TEXT" NAME="itemNum"><BR>
  Quantity: <INPUT TYPE="TEXT" NAME="quantity"><BR>
  Price Each: <INPUT TYPE="TEXT" NAME="price" VALUE="$"><BR>
  First Name: <INPUT TYPE="TEXT" NAME="firstName"><BR>
  <TEXTAREA NAME="address" ROWS=3 COLS=40></TEXTAREA><BR>
  Credit Card Number:
  <INPUT TYPE="PASSWORD" NAME="cardNum"><BR>
  <CENTER>
    <INPUT TYPE="SUBMIT" VALUE="Submit Order">
  </CENTER>
</FORM>
</BODY>
</HTML>
```

A Sample FORM using POST



A Form Based Servlet: POST

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

public class ShowParameters extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
                    throws ServletException, IOException {
    ...
  }
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
                    throws ServletException, IOException {
    doGet(request, response);
  }
}
```

Who Set Object/value Attributes

- Request attributes can be set in two ways
  - Servlet container itself might set attributes to make available custom information about a request
    - example: javax.servlet.request.X509Certificate attribute for HTTPS
  - Servlet set application-specific attribute
    - *void setAttribute(java.lang.String name, java.lang.Object o)*

Getting Locale Information

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

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

      if (messages == null) {
         Locale locale=request.getLocale();
         messages = ResourceBundle.getBundle(
                "messages.BookstoreMessages", locale);
         session.setAttribute("messages", messages);
      }
```

Getting Client 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

Getting Server Information

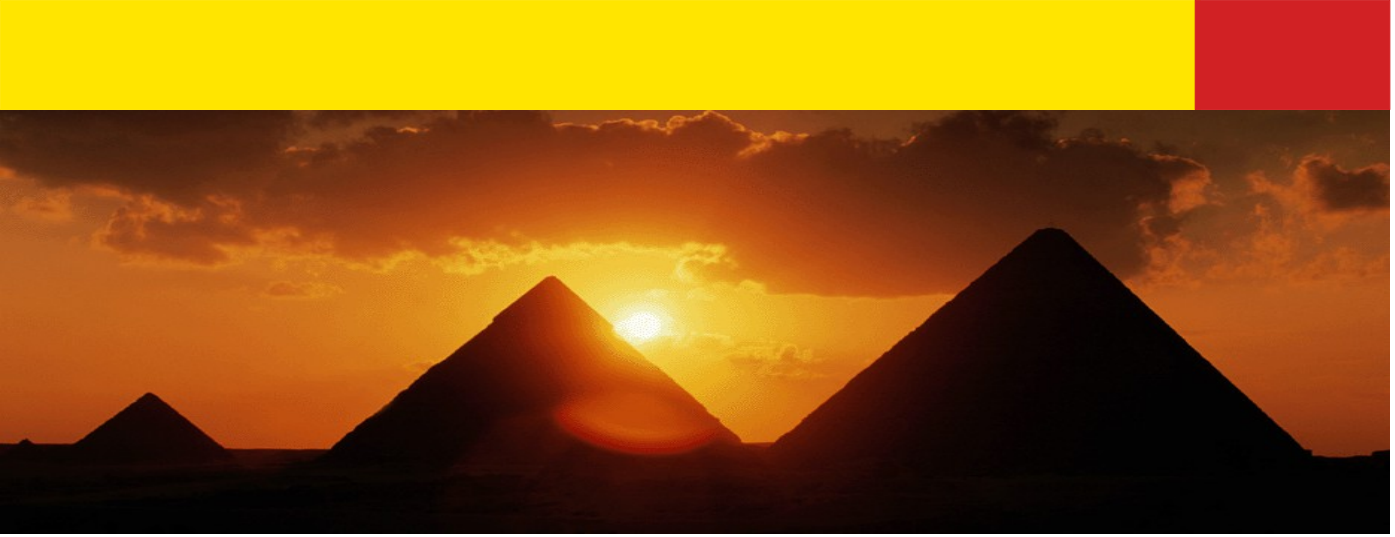
- Servlet can get server's information:
  - String request.getServerName()
    - e.g. "www.sun.com"
  - int request.getServerPort()
    - e.g. Port number "8080"

Getting Misc. Information

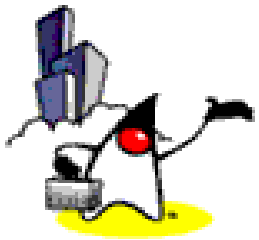
- Input stream
  - ServletInputStream getInputStream()
  - java.io.BufferedReader getReader()
- Protocol
  - java.lang.String getProtocol()
- Content type
  - java.lang.String getContentType()
- Is secure or not (if it is HTTPS or not)
  - boolean isSecure()

Cookie Method (in HTTPServletRequest)

- Cookie[] getCookies()
  - Returns an array containing all of the Cookie objects the client sent with this request



HTTP Request URL



HTTP Request URL

- Contains the following parts
  - http://[host]:[port]/[request path]?[query string]

HTTP Request URL: [request path]

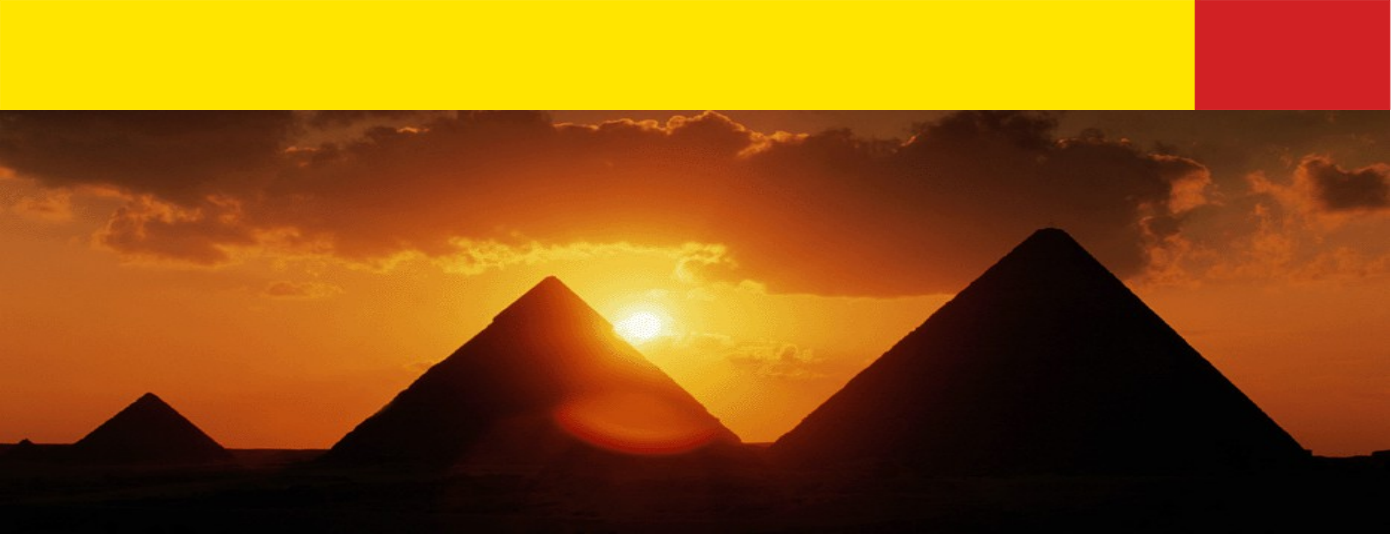
- http://[host]:[port]/[request path]?[query string]
- [request path] is made of
  - Context: /<context of web app>
  - Servlet name: /<component alias>
  - Path information: the rest of it
- Examples
  - http://localhost:8080/hello1/greeting
  - http://localhost:8080/hello1/greeting.jsp
  - http://daydreamer/catalog/lawn/xyz.jsp

HTTP Request URL: [query string]

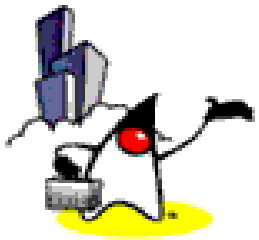
- http://[host]:[port]/[request path]?[query string]
- [query string] are composed of a set of parameters and values that are user entered
- Two ways query strings are generated
  - A query string can explicitly appear in a web page
    - Add To Cart
    - String bookId = request.getParameter("Add");
  - A query string is appended to a URL when a form with a GET HTTP method is submitted
    - http://localhost/hello1/greeting?username=Monica
    - String userName=request.getParameter("username")

Context, Path, Query, Parameter Methods

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



HTTP Request Headers



HTTP Request Headers

- HTTP requests include headers which provide extra information about the request
- Example of HTTP 1.1 Request:

```
GET /search? keywords= servlets+ jsp HTTP/ 1.1
Accept:  image/ gif, image/ jpg, */*
Accept-Encoding: gzip
Connection:  Keep- Alive
Cookie:  userID= id456578
Host:  www.sun.com
Referer:  http:/www.jpassion.com/codecamp.html
User-Agent:  Mozilla/ 4.7 [en] (Win98; U)
```

HTTP Request Headers

- Accept
  - Indicates MIME types browser can handle.
- Accept-Encoding
  - Indicates encoding (e. g., gzip or compress) browser can handle
- Etc.

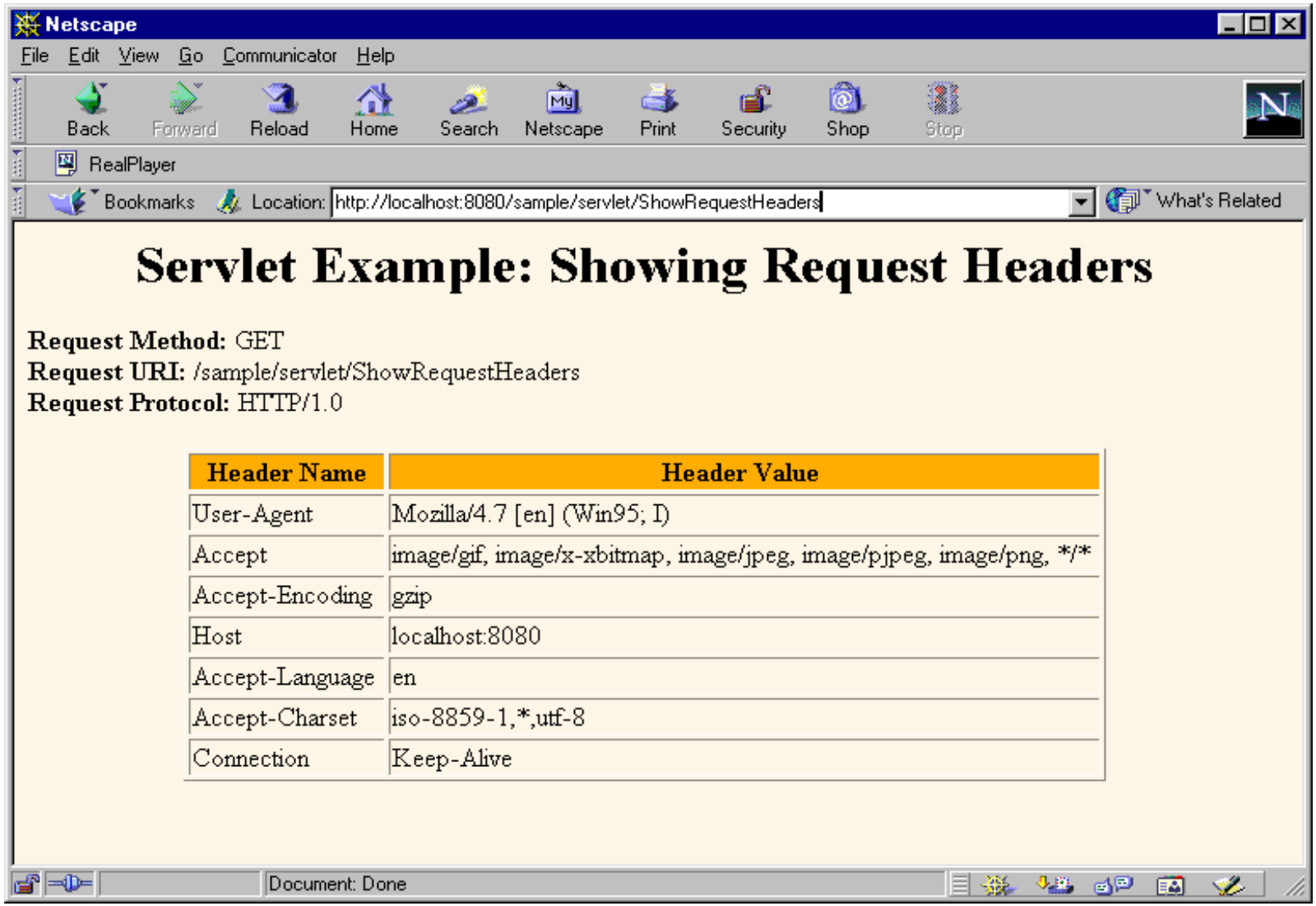
HTTP Header Methods

- String getHeader(java.lang.String name)
  - value of the specified request header as String
- java.util.Enumeration getHeaders(java.lang.String name)
  - values of the specified request header
- java.util.Enumeration getHeaderNames()
  - names of request headers
- int getIntHeader(java.lang.String name)
  - value of the specified request header as an int

Showing Request Headers

```
//Shows all the request headers sent on this particular request.
public class ShowRequestHeaders extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
                    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Servlet Example: Showing Request Headers";
    out.println("<HTML>" + ...
                "<B>Request Method: </B>" +
                request.getMethod() + "<BR>\n" +
                "<B>Request URI: </B>" +
                request.getRequestURI() + "<BR>\n" +
                "<B>Request Protocol: </B>" +
                request.getProtocol() + "<BR><BR>\n" +
            ...
                "<TH>Header Name<TH>Header Value");
    Enumeration headerNames = request.getHeaderNames();
    while(headerNames.hasMoreElements()) {
      String headerName = (String)headerNames.nextElement();
      out.println("<TR><TD>" + headerName);
      out.println("    <TD>" + request.getHeader(headerName));
    }
    ...
  }
}
```

Request Headers Sample



Code with Passion!
JPassion.com