Servlet

Contents

- Servlet introduction
- Servlet request & response model
- Servlet life cycle
- Scope objects
- Including & forwarding ro another web resource
- Servlet filters
- Servlets life cycle events
- Synchronization & thread Model
- Handling Errors

Servlet Introduction

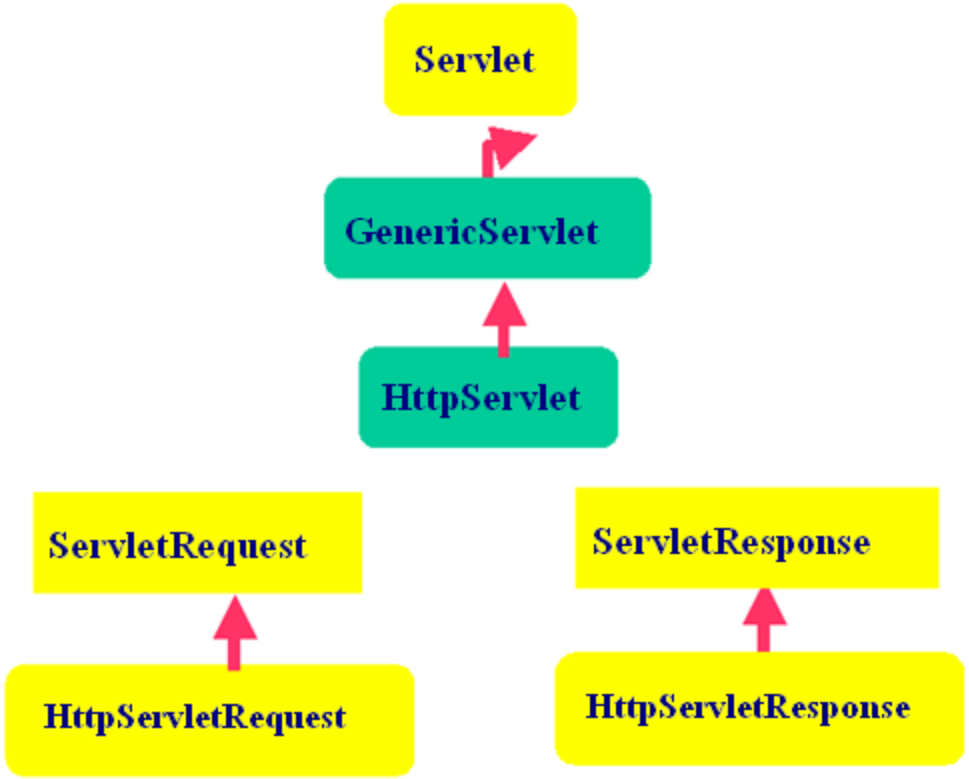
What is Servlet?

- **Java Servlet** A Java program that extends the functionality of a Web server, generating dynamic content and interacting with Web clients using a **request-response paradigm**.
- Platform and server independent

Servlet

- The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets.
- All servlets must implement the Servlet interface, which defines life-cycle methods.
- When implementing a generic service, use or extend the GenericServlet class.
  - Defines a generic, protocol-independent servlet.
  - implements the Servlet and ServletConfig interfaces.
  - GenericServlet makes writing servlets easier.
- The HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services.
  - Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site.

Servlet Interfaces & Classes



First Servlet Code

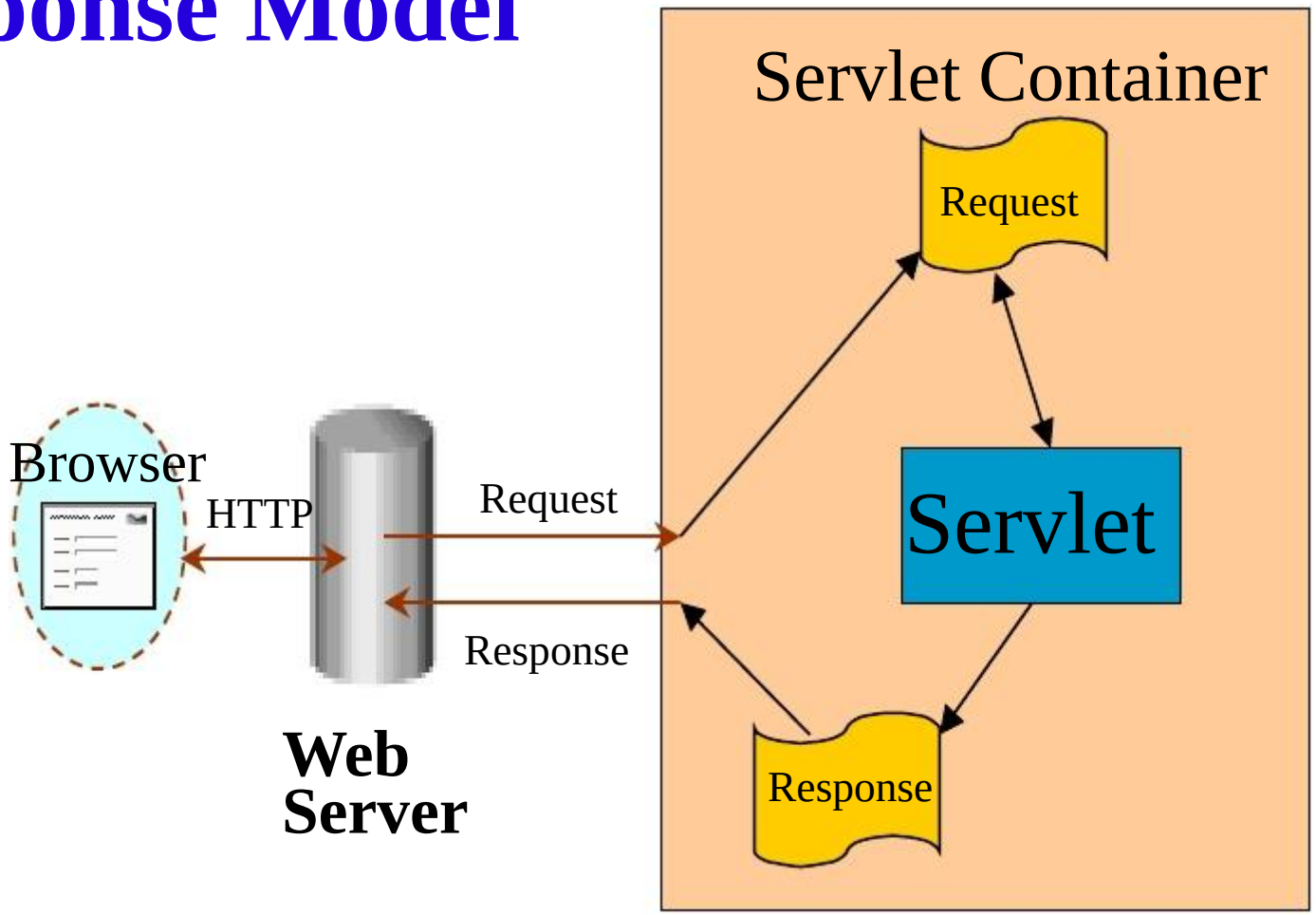
```
Public class HelloServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
                                        HttpServletResponse
response){
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<title>Hello World!</title>");
    }
    ...
}
```

Servlet Request & Response Model

Container

- A **container** is nothing but a piece of software responsible for loading, initializing, executing and unloading the Servlets and JSP.

Servlet Request and Response Model



What does Servlet Do?

- Receives client request (mostly in the form of HTTP request)
- Extract some information from the request
- Do content generation or business logic process (possibly by accessing database, invoking EJBs, etc)
- Create and send response to client (mostly in the form of HTTP response) or forward the request to another servlet or JSP page

HTTP GET and POST

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

Servlet

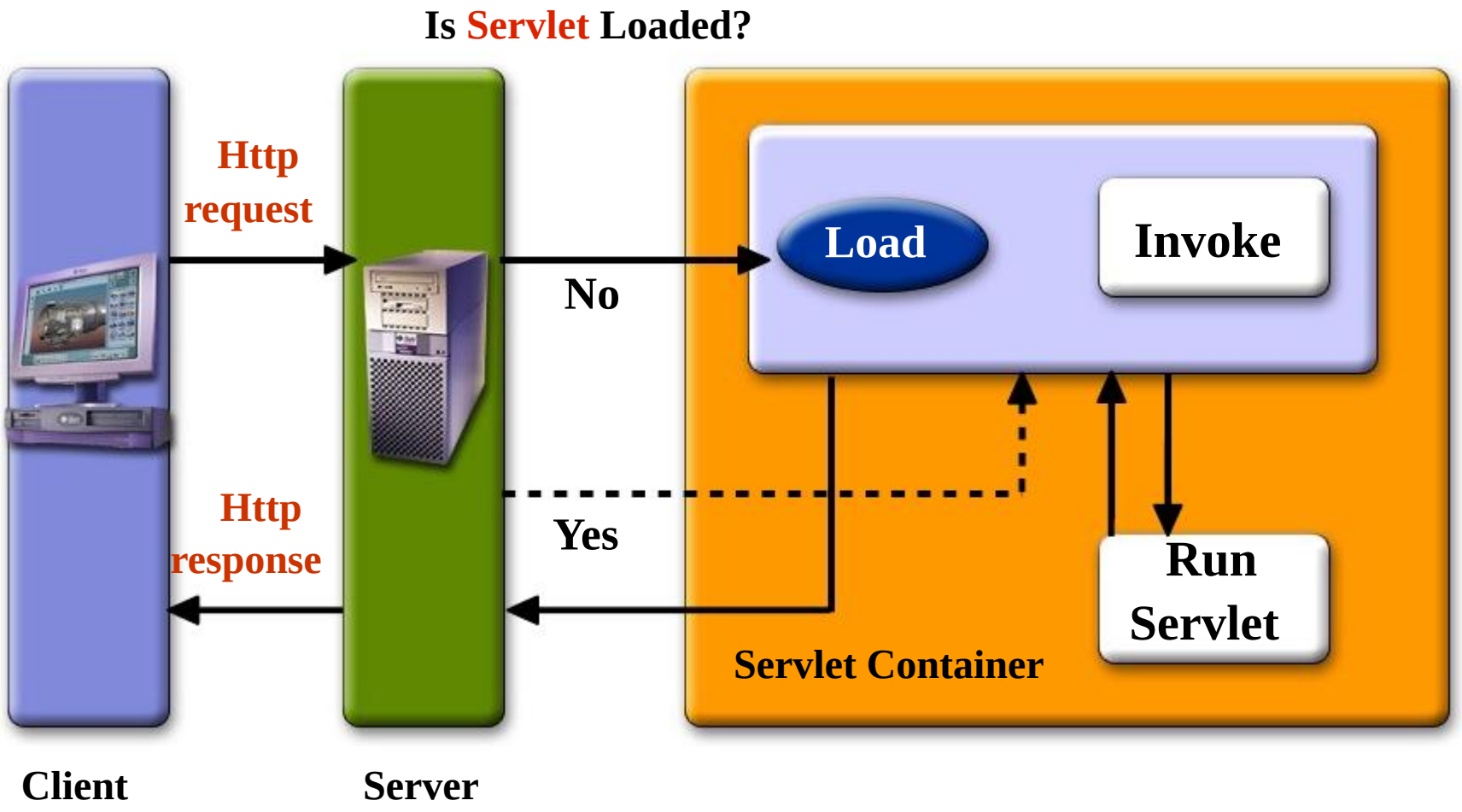
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

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>");
    }
}
```

Servlet Life-Cycle

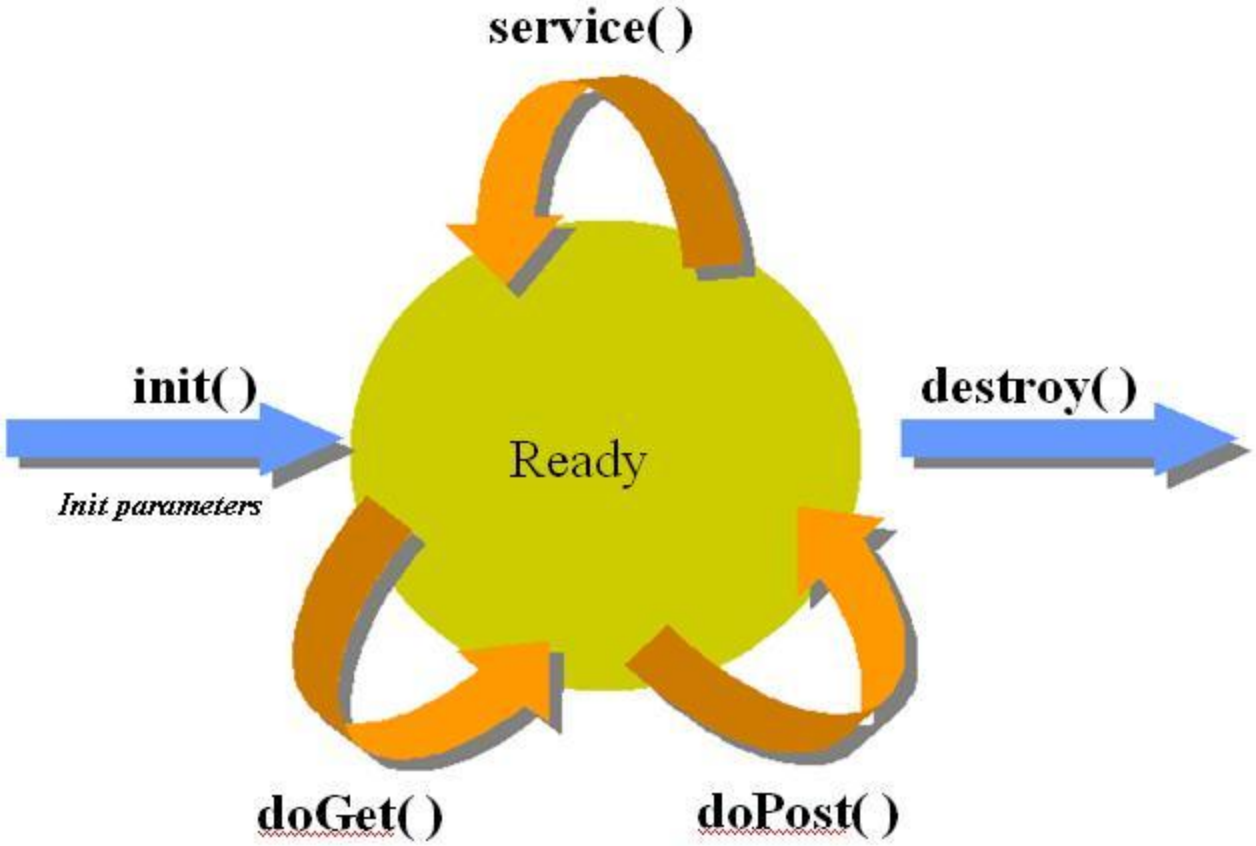
Servlet Life-Cycle

- **The life cycle of a servlet is controlled by the container.**
- **When a request is mapped to a servlet, the container performs the following steps:**
  - **If an instance of the servlet does not exist, the web container**
    - **Loads the servlet class.**
    - **Creates an instance of the servlet class.**
    - **Initializes the servlet instance by calling the init method.**
  - **Invokes the service method, passing request and response objects.**
  - **If the container needs to remove the servlet, it finalizes the servlet by calling the servlet's destroy method.**

Servlet Life-Cycle



Servlet 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
    - doGet(), doPost(), doXxx()
    - service() - implementation

Servlet Life Cycle Methods

- init()
  public void init(ServletConfig config) throws ServletException
- The init() method is invoked only once by the servlet container throughout the life of a servlet.
- Perform any set-up in this method
  - Setting up a database connection
- a ServletConfig object contains the initialization parameters and servlet's configuration.
- By this init() method the servlet get to know that it has been placed into service.
- The servlet cannot be put into the service if
  - The init() method does not return within a fix time set by the web server.
  - It throws a ServletException

Servlet Life Cycle Methods

- destroy()
  - called when closing the servlet.
  - Invoked before servlet instance is removed
  - Perform any clean-up
    - Closing a previously created database connection
    - Release all the resources like memory, threads etc

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>
```

Example: destroy()

```
public class CatalogServlet extends HttpServlet {
        private BookDB bookDB;


        public void init() throws ServletException {
                bookDB = (BookDB)getServletContext().
                                                getAttribute("bookDB");
                if (bookDB == null) throw new
                        UnavailableException("Couldn't get database.");
        }
        public void destroy() {
                                        bookDB = null;
        }
        ...
}
```

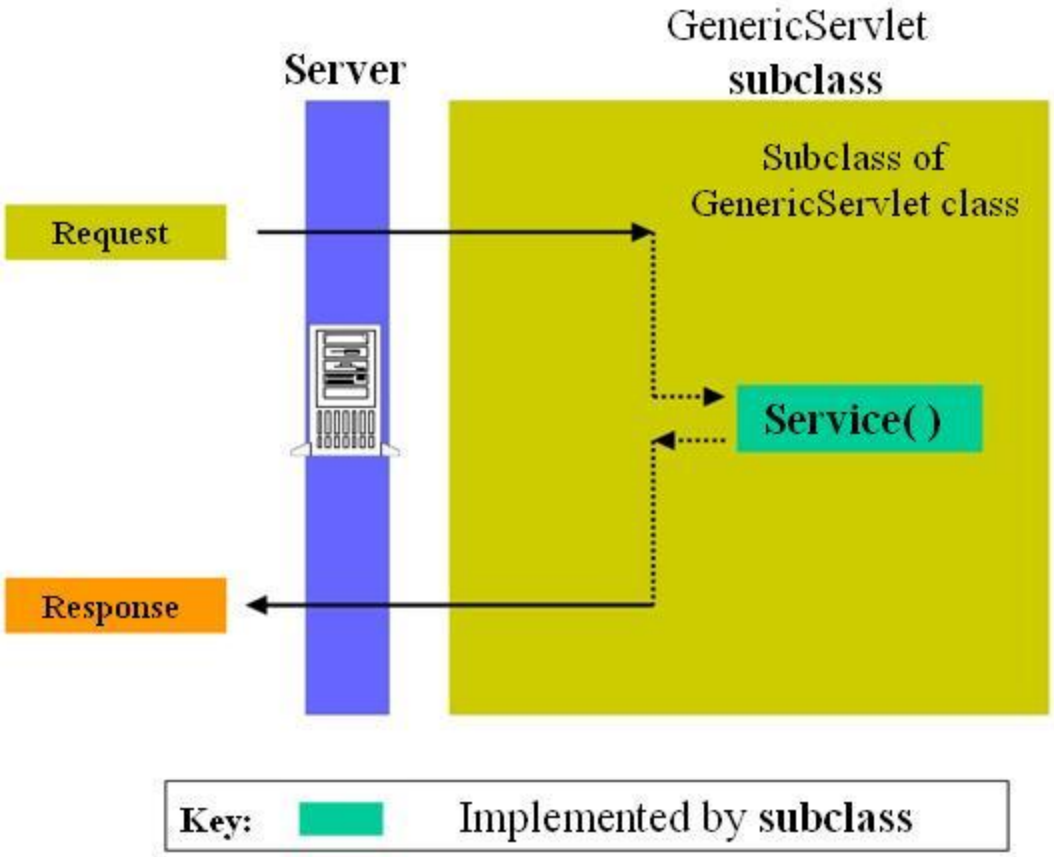
Servlet Life Cycle Methods

- 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 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

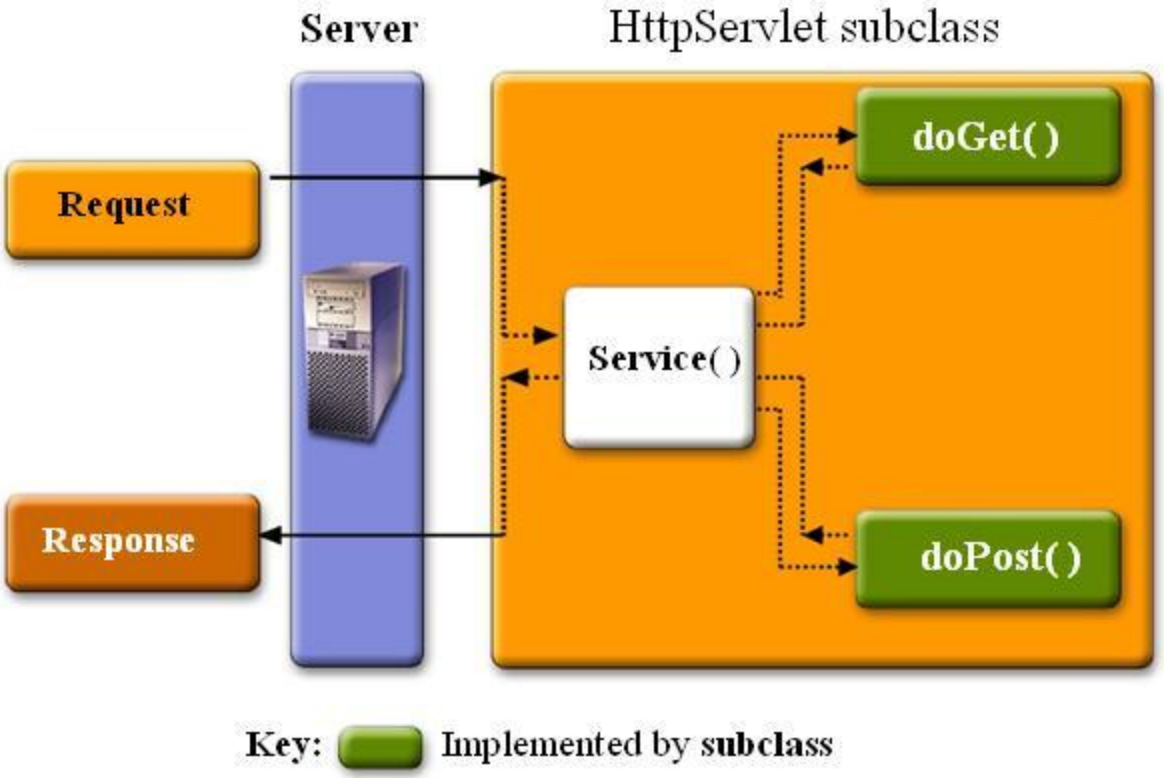
service() & doGet()/doPost()

- service() methods take generic requests and responses:
  - public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException
- doGet() or doPost() take HTTP requests and responses:
  - protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, java.io.IOException
  - protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, java.io.IOException

Service() Method



doGet() and doPost() Methods



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: 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 “message”
        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 the identifier of the book to display (Get HTTP parameter)
        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: 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 {
            // 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>");
    }
}
```

Scope Objects

Scope 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

Four Scope Objects: Accessibility

- Web context (ServletConext)
  - Shared by all web components within a single web application
  - Accessible from Web components within a Web context
- Session
  - Shared by web components that share a same session
  - Accessible from Web components handling a request that belongs to the session
- Request
  - Shared by web components that handle the same request
  - Accessible from Web components handling the same Request
- Page
  - Used within a JSP page
  - Accessible from JSP page that creates the object

Four Scope Objects: Class

- Web context
  - javax.servlet.ServletContext
- Session
  - javax.servlet.http.HttpSession
- Request
  - subtype of javax.servlet.ServletRequest: javax.servlet.http.HttpServletRequest
- Page
  - javax.servlet.jsp.PageContext

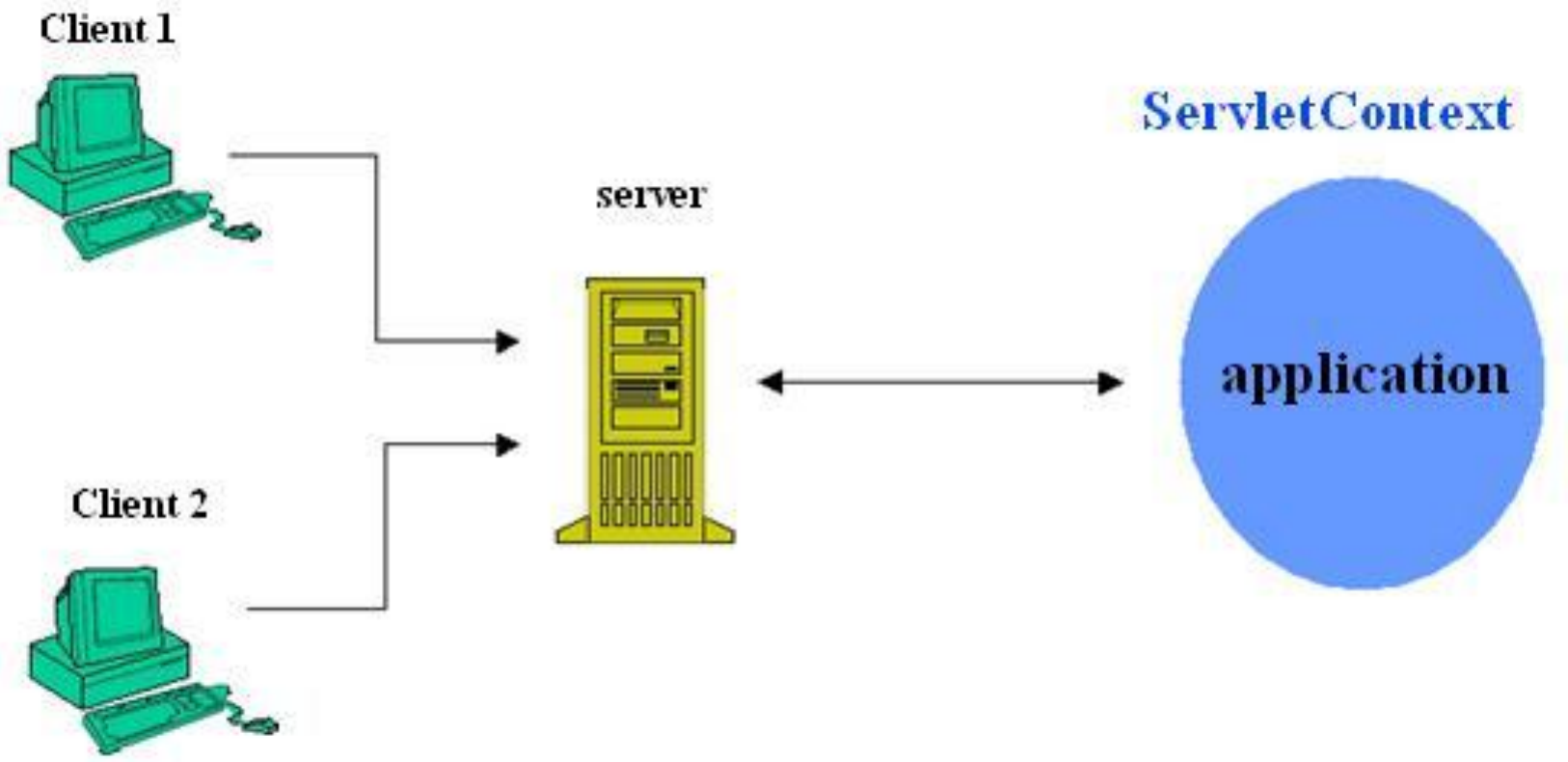
What is ServletContext For?

- Used by servlets to
  - Set and get context-wide (application-wide) object-valued attributes
  - Get request dispatcher
    - To forward to or include web component
  - Access Web context-wide initialization parameters set in the web.xml file
  - Access Web resources associated with the Web context
  - Log
  - Access other information

Scope of ServletContext

- Context-wide scope
  - Shared by all servlets and JSP pages within a "web application"
    - Why it is called “web application scope”
  - A "web application" is a collection of servlets and content installed under a specific subset of the server's URL namespace and possibly installed via a *.war file
  - There is one ServletContext object per "web application"

ServletContext: Web Application Scope



How to Access ServletContext Object?

- Within your servlet code, call getServletContext()
- Within your servlet filter code, call getServletContext()
- The ServletContext is contained in ServletConfig object, which the Web server provides to a servlet when the servlet is initialized
  - init (ServletConfig servletConfig) in Servlet interface

Example: Getting Attribute Value from ServletContext

```
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: Getting and Using RequestDispatcher Object

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

Example: Logging

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

        ...
        getServletContext().log("Life is good!");
        ...
        getServletContext().log("Life is bad!", someException);
```

HttpSession

Why HttpSession?

- Need a mechanism to maintain client state across a series of requests from the same user (or originating from the same browser) over some period of time
  - Example: Online shopping cart
- HTTP is stateless
- HttpSession maintains client state
  - Used by Servlets to set and get the values of session scope attributes

How to Get HttpSession?

- via getSession() method of a Request object (HttpServletRequest)

Example: HttpSession

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

        // Get the user's session and shopping cart
        HttpSession session = request.getSession();
        ShoppingCart cart =
            (ShoppingCart)session.getAttribute("cart");
        ...
        // Determine the total price of the user's books
        double total = cart.getTotal();
```

Including another Web Resource

When to Include another Web resource?

- When it is useful to add static or dynamic contents already created by another web resource
  - a copyright information

Types of Included Web Resource

- Static resource
  - It is like “programmatic” way of adding the static contents in the response of the “including” servlet
- 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

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

How to Include another Web resource?

- Get RequestDispatcher object from ServletConext 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

```
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>  " +
                "<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>   <hr><br> ");
    }
public void doPost (HttpServletRequest request, HttpServletResponse response)
                                    throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("<body bgcolor=\"#ffffff\">" + "<center>" + "<hr> <br>  " +
                "<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>   <hr> <br> ");
}
}
```

Example

```
out.println(  "<html>" + "<head><title>"  + messages.getString("TitleBookCatalog") + "</title></head>");
      // Get the dispatcher; it gets the banner to the user
      RequestDispatcher dispatcher = getServletContext() .getRequestDispatcher("/banner");
      if (dispatcher != null) {    dispatcher.include(request, response);       }
        // Additions to the shopping cart
      String bookId = request.getParameter("bookId");
      if (bookId != null) {
          try {
              Book book = bookDB.getBook(bookId);
              cart.add(bookId, book);
              out.println( "<p><h3>" + "<font color=\"#ff0000\">" + messages.getString("CartAdded1") +
    "<i>" + book.getTitle() + "</i> "
                      + messages.getString("CartAdded2") + "</font></h3>");
          } catch (BookNotFoundException ex) {
              response.reset();
              throw new ServletException(ex);
          }
      }
      //Give the option of checking cart or checking out if cart not empty
      if (cart.getNumberOfItems() > 0) {
          out.println( "<p><strong><a href=\""  + response.encodeURL(request.getContextPath() +
    "/bookshowcart") + "\">" + messages.getString("CartCheck")
                  + "</a>   " + "<a href=\"" +
    response.encodeURL(request.getContextPath() + "/bookcashier") + "\">"
                  + messages.getString("Buy") + "</a>" + "</p></strong>");
      }
```

Forwarding to another Web Resource

When to use “Forwarding” to another Web resource?

- When you want to have one Web component do preliminary processing of a request and have another component generate the response

Rules of “Forwarding” to another Web resource?

- Should be used to give another resource responsibility for replying to the user
  - If you have already accessed a ServletOutputStream or PrintWriter object within the servlet, you cannot use this method; it throws an IllegalStateException

How to do “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,
  ...
}
```

Servlet Filters

What are Java Servlet Filters?

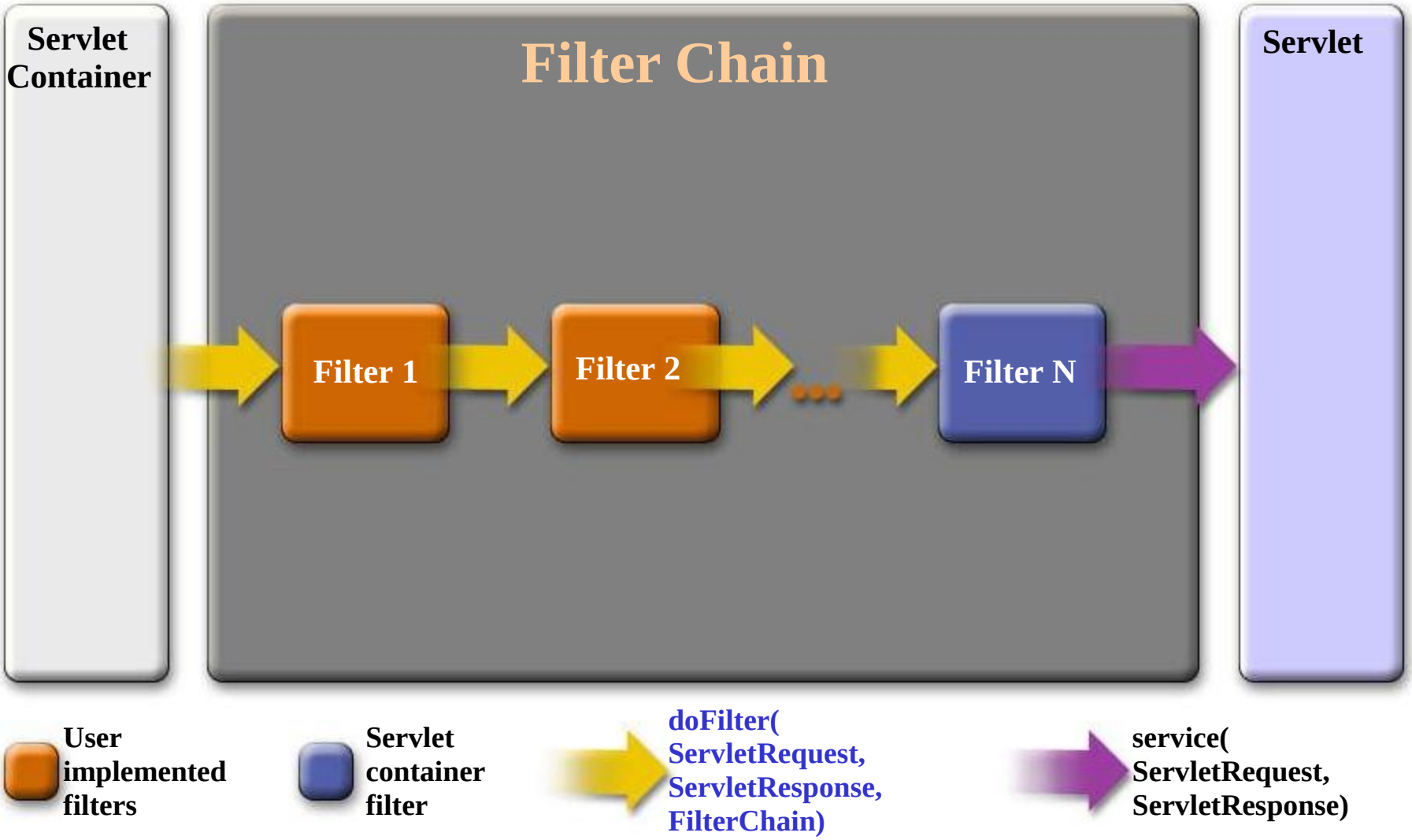
- New component framework for intercepting and modifying requests and responses
  - Filters can be chained and plugged in to the system during deployment time
- Allows range of custom activities:
  - Marking access, blocking access
  - Caching, compression, logging
  - Authentication, access control, encryption
  - Content transformations
- Introduced in Servlet 2.3 (Tomcat 4.0)

What Can a Filter Do?

- Customize the request object if it wishes to modify request headers or data
- Customize the response object if it wishes to modify response headers or data
- Invoke the next entity in the filter chain
- Examine response headers after it has invoked the next filter in the chain
- Throw an exception to indicate an error in processing

Servlet Filter Chain

How Servlet Filter Work?



How Filter Chain Works

- Multiple filters can be chained
  - order is dictated by the order of <filter> elements in the web.xml deployment descriptor
- The first filter of the filter chain is invoked by the container
  - via doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
  - the filter then perform whatever filter logic and then call the next filter in the chain by calling chain.doFilter(..) method
- The last filter's call to chain.doFilter() ends up calling service() method of the Servlet

Servlet Filter Programming APIs

javax.servlet.Filter Interface

- init(FilterConfig)
  - called only once when the filter is first initialized
  - get ServletContext object from FilterConfig object
  - read filter initialization parameters from FilterConfig object through getInitParameter() method
- destroy()
  - called only once when container removes filter object
  - close files or database connections

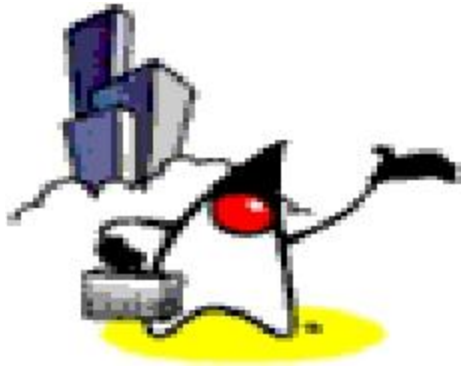
javax.servlet.Filter Interface

- doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
  - gets called each time a filter is invoked
  - contains most of filtering logic
  - ServletRequest object is casted to HttpServletRequest if the
  - request is HTTP request type
  - may wrap request/response objects
  - invoke next filter by calling chain.doFilter(..)
  - or block request processing
    - by omitting calling chain.doFilter(..)
    - filter has to provide output response to the client
  - set headers on the response for next entity

Other Sevlet Filter Related Classes

- javax.servlet.FilterChain
  - passed as a parameter in doFilter() method
- javax.servlet.FilterConfig
  - passed as a parameter in init() method
- javax.servlet.HttpServletResponseWrapper
  - convenient implementation of the HttpServletResponse interface

Servlet Filter Configuration in the web.xml file



Configuration in web.xml

- <filter>
  - <filter-name>: assigns a name of your choosing to the filter
  - <filter-class>: used by the container to identify the filter class
- </filter>
- <filter-mapping>
  - <filter-name>: assigns a name of your choosing to the filter
  - <url-pattern>: declares a pattern URLs (Web resources) to which the filter applies
- </filter-mapping>

Example

```
<web-app>
        <display-name>Bookstore1</display-name>
        <description>no description</description>
        <filter>
                <filter-name>OrderFilter</filter-name>
                <filter-class>filters.OrderFilter</filter-class>
        </filter>
        <filter>
                <filter-name>HitCounterFilter</filter-name>
                <filter-class>filters.HitCounterFilter</filter-class>
        </filter>
        <filter-mapping>
                <filter-name>OrderFilter</filter-name>
                <url-pattern>/receipt</url-pattern>
        </filter-mapping>
        <filter-mapping>
                <filter-name>HitCounterFilter</filter-name>
                <url-pattern>/enter</url-pattern>
        </filter-mapping>
        <listener>
        ...
        </listener>
        <servlet>
        ...
```

Steps for Building a Servlet Filter

- Decide what custom filtering behavior you want to implement for a web resource
- Create a class that implements Filter interface
  - Implement filtering logic in the doFilter() method
  - Call the doFilter() method of FilterChain object
- Configure the filter
  - use <filter> and <filter-mapping> elements

Example: HitCounterFilter

```
public final class HitCounterFilter implements Filter {
    private FilterConfig filterConfig = null;

    public void init(FilterConfig filterConfig)
        throws ServletException {
        this.filterConfig = filterConfig;
    }
    public void destroy() {
        this.filterConfig = null;
    }

    // Continued in the next page...
```

Example: HitCounterFilter

```
public void doFilter(ServletRequest request,
                     ServletResponse response, FilterChain chain)
                     throws IOException, ServletException {

    if (filterConfig == null) return;
    StringWriter sw = new StringWriter();
    PrintWriter writer = new PrintWriter(sw);
    Counter counter =
      (Counter) filterConfig.getServletContext().getAttribute("hitCounter");
    writer.println("The number of hits is: " +
        counter.incCounter());

    // Log the resulting string
    writer.flush();
    filterConfig.getServletContext().log(sw.getBuffer().toString());
    ...
    chain.doFilter(request, wrapper);
    ...
  }
}
```

HitCounterFilter Configuration

```
<?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>Bookstore1</display-name>
   <description>no description</description>

   <filter>
      <filter-name>HitCounterFilter</filter-name>
      <filter-class>filters.HitCounterFilter</filter-class>
   </filter>

   <filter-mapping>
      <filter-name>HitCounterFilter</filter-name>
      <url-pattern>/enter</url-pattern>
   </filter-mapping>
   ...
```

Servlet LifeCycle Events

Servlet Lifecycle Events

- Support event notifications for state changes in
  - ServletContext
    - Startup/shutdown
    - Attribute changes
  - HttpSession
    - Creation and invalidation
    - Changes in attributes

Steps for Implementing Servlet Lifecycle Event

1. Decide which scope object you want to receive an event notification
2. Implement appropriate interface
3. Override methods that need to respond to the events of interest
4. Obtain access to important Web application objects and use them
5. Configure web.xml accordingly
6. Provide any needed initialization parameters

Listener Registration

- Web container
  - creates an instance of each listener class
  - registers it for event notifications.
  - Registers the listener instances according to
    - the interfaces they implement
    - the order in which they appear in the deployment descriptor web.xml
- Listeners are invoked in the order of their registration during execution

Listener Interfaces

- ServletContextListener
  - contextInitialized/Destroyed(ServletContextEvent)
- ServletContextAttributeListener
  - attributeAdded/Removed/Replaced(ServletContextAttributeEvent)
- HttpSessionListener
  - sessionCreated/Destroyed(HttpSessionEvent)
- HttpSessionAttributeListener
  - attributedAdded/Removed/Replaced(HttpSessionBindingEvent)
- HttpSessionActivationListener
  - sessionWillPassivate(HttpSessionEvent)
  - sessionDidActivate(HttpSessionEvent)

Example: Context Listener

```
public final class ContextListener
        implements ServletContextListener {
        private ServletContext context = null;

        public void contextInitialized(ServletContextEvent event) {
                context = event.getServletContext();

                try {
                        BookDB bookDB = new BookDB();
                        context.setAttribute("bookDB", bookDB);
                } catch (Exception ex) {
                        context.log("Couldn't create bookstore
                                                database bean: " + ex.getMessage());
                }

                Counter counter = new Counter();
                context.setAttribute("hitCounter", counter);

        }
```

Example: Context Listener

```
    public void contextDestroyed(ServletContextEvent event) {
            context = event.getServletContext();
            BookDB bookDB = (BookDB)context.getAttribute
("bookDB");
            bookDB.remove();
            context.removeAttribute("bookDB");
            context.removeAttribute("hitCounter");

    }
}
```

Listener Configuration

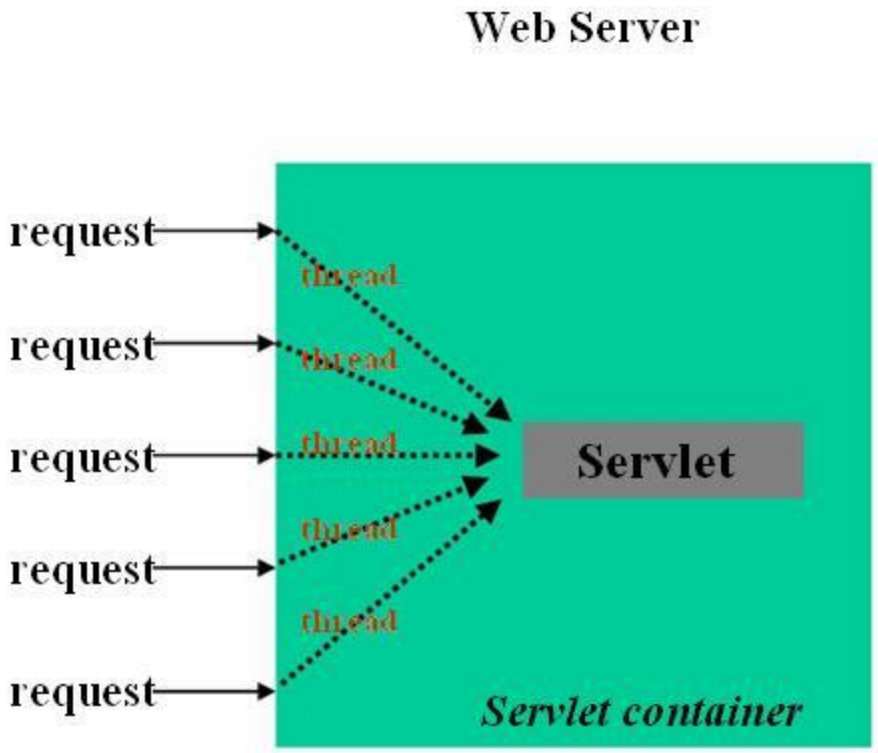
```
<web-app>
    <display-name>Bookstore1</display-name>
    <description>no description</description>

    <filter>..</filter>
    <filter-mapping>..</filter-mapping>
    <listener>
        <listener-class>listeners.ContextListener</listener-class>
    </listener>
    <servlet>..</servlet>
    <servlet-mapping>..</servlet-mapping>
    <session-config>..</session-config>
    <error-page>..</error-page>
    ...
</web-app>
```

Synchronization & Thread Model

Concurrency Issues on a Servlet

- The service() method of a servlet instance can be invoked by multiple clients (multiple threads)
- Servlet programmer has to deal with concurrency issue
  - shared data needs to be protected
  - this is called "servlet synchronization"
- 2 options for servlet synchronization
  - use of synchronized block
  - use of SingleThreadModel

Many Threads, One Servlet Instance



Use of synchronized block

- Synchronized blocks are used to guarantee only one thread at a time can execute within a section of code

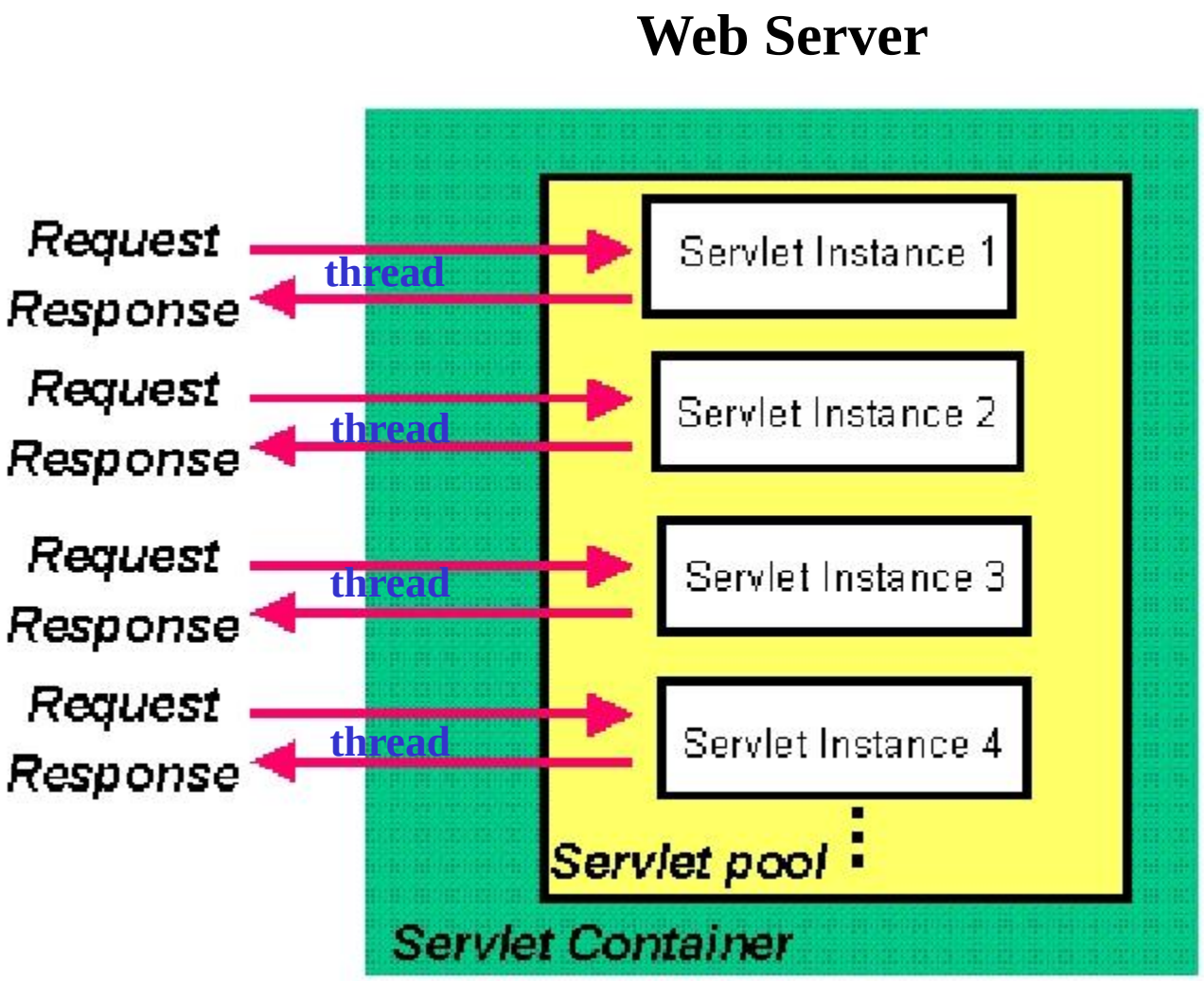
```
synchronized(this) {
        myNumber = counter + 1;
        counter = myNumber;
}
...
synchronized(this) {
        counter = counter - 1 ;
}
```

SingleThreadModel Interface

- Servlets can also implement javax.servlet.SingleThreadModel
  - Ensures that servlets handle only one request at a time.
- The server will manage a pool of servlet instances and dispatching each new request to a free servlet.
- Guaranteed there will only be one thread per instance
- This could be overkill in many instances

```
Public class SingleThreadModelServlet extends
        HttpServlet implements SingleThreadModel {
        ...
}
```

SingleThreadModel



Best Practice Recommendation

- Do use synchronized block whenever possible
  - SingleThreadModel is expensive (performance wise)

Handling Errors

Handling Errors

- Web container generates default error page
- You can specify custom default page to be displayed instead
- Steps to handle errors
  - Create appropriate error html pages for error conditions
  - Modify the web.xml accordingly

Example: Setting Error Pages in web.xml

```
<error-page>
    <exception-type>
        exception.BookNotFoundException
    </exception-type>
    <location>/errorpage1.html</location>
</error-page>
<error-page>
    <exception-type>
        exception.BooksNotFoundException
    </exception-type>
    <location>/errorpage2.html</location>
</error-page>
<error-page>
    <exception-type>exception.OrderException</exception-type>
    <location>/errorpage3.html</location>
</error-page>
```

The End