CSBP 461
Internet Computing:

Java Servlets (Part1)

Dr. M. Elarbi Badidi

Topics

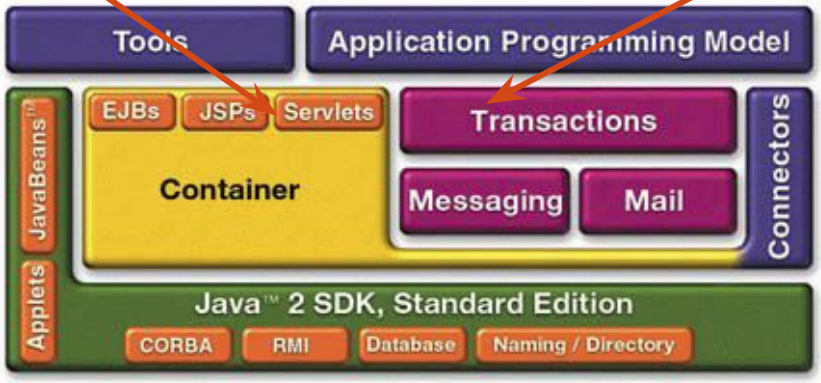
- Servlet in big picture of J2EE
- Servlet request & response model
- Servlet life cycle
- Servlet scope objects
- Servlet request
- Servlet response: Status, Header, Body

SERVLET IN A
BIG PICTURE OF J2EE

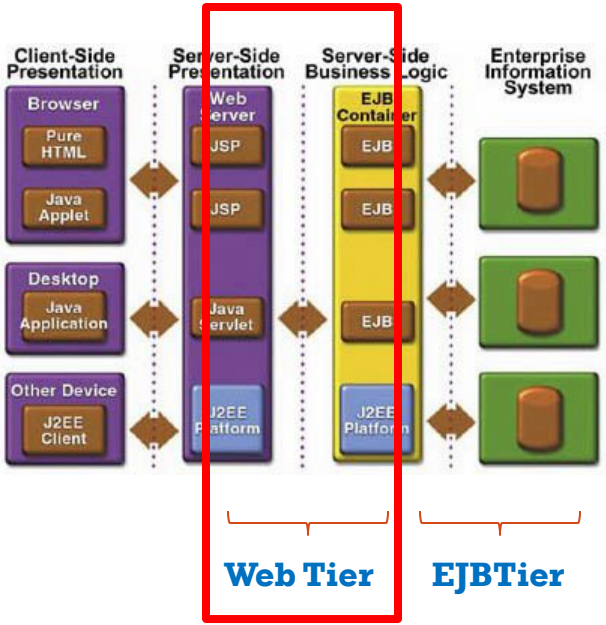
J2EE Architecture

- An extensible Web technology that uses template data, custom elements, scripting languages, and server-side Java objects to return dynamic content to a client. Typically the template data is HTML or XML elements. The client is often a Web browser.

- 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.



Where are Servlet and JSP?



Container

- The Servlet container is responsible of managing the life cycle of a Servlet object. Functions of the Servlet container include:
  - taking input requests from clients;
  - instantiating an instance of the Servlet class;
  - passing the requests to the Servlet object and letting the Servlet object process the requests;
  - forwarding the results to clients.

Container

- A Java Servlet application is supported by its Servlet container. *The container may be an add-on Servlet container or standalone container, which comes as a part of a Web server.*

- Since a Java Servlet itself is a Java class. It needs Java API support, specifically the Java Servlet API that is available in an archive file called *servlet-api.jar* in Tomcat.

Container

- The Apache Tomcat web server is the official reference implementation of Servlet containers, supporting Servlets and JSP.
- The Tomcat Web server is an open source Servlet container originally developed by Sun Microsystems.
- There are many other Web servers supporting Servlets and JSP, such as *Sun's Java Web server*, and *Macromedia's JRun*, *Caucho Resin*, and *Jetty*.
- Many application servers, like *Sun Java System Application Server*, *BEA WebLogic* and *IBM WebSphere*, *Oracle Application Server*, *Pramati Server*, *JBoss* also support Servlets and JSP.

What is Servlet?

- Java™ objects which are based on servlet framework and APIs and extend the functionality of a HTTP server.
- 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

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

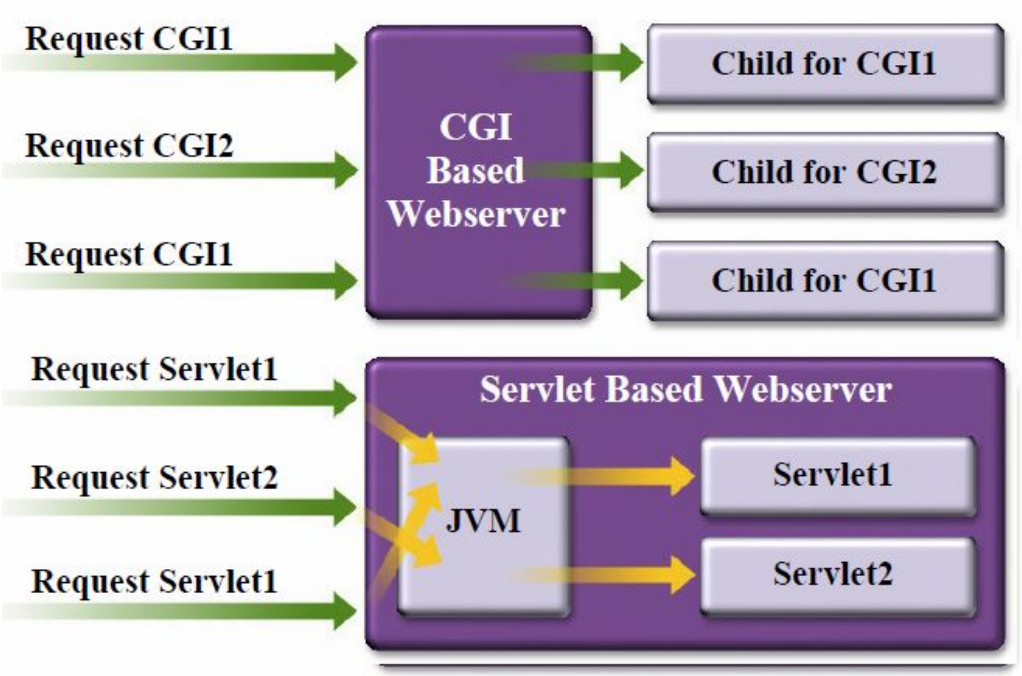
CGI

- 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

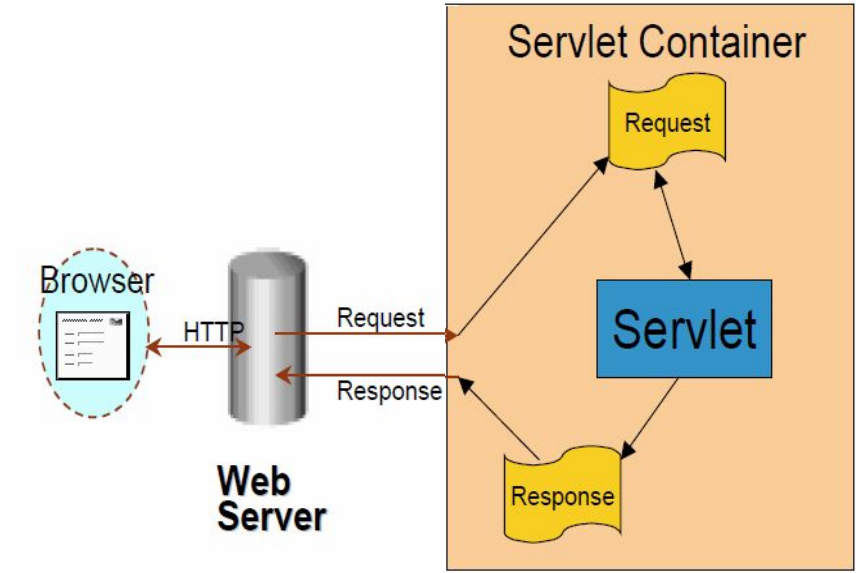


Advantages of Servlet

- 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 REQUEST & RESPONSE MODEL

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

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 method
    - Get: Input form data is passed as part of URL
    - Post: Input form data is passed within message body
    - Put
    - Header
- request data

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

First 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>");
 }
}
```

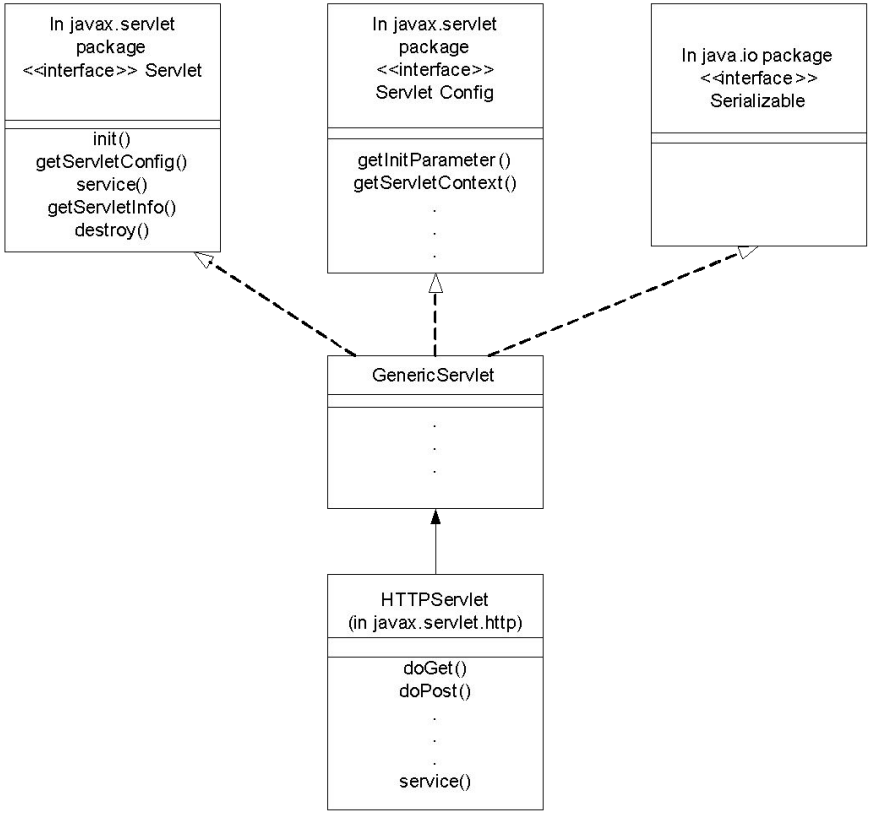
INTERFACES & CLASSES OF SERVLET

Servlet Interfaces & Classes

- A Java *Servlet* is just a typical Java class which extends an abstract class *HttpServlet*.

- The *HttpServlet* class extends another abstract class *GenericServlet* . The *GenericServlet* class implements three interfaces:
  - *javax.servlet.Servlet*,
  - *javax.servlet.ServletConfig*, and
  - *java.io.Serializable*.

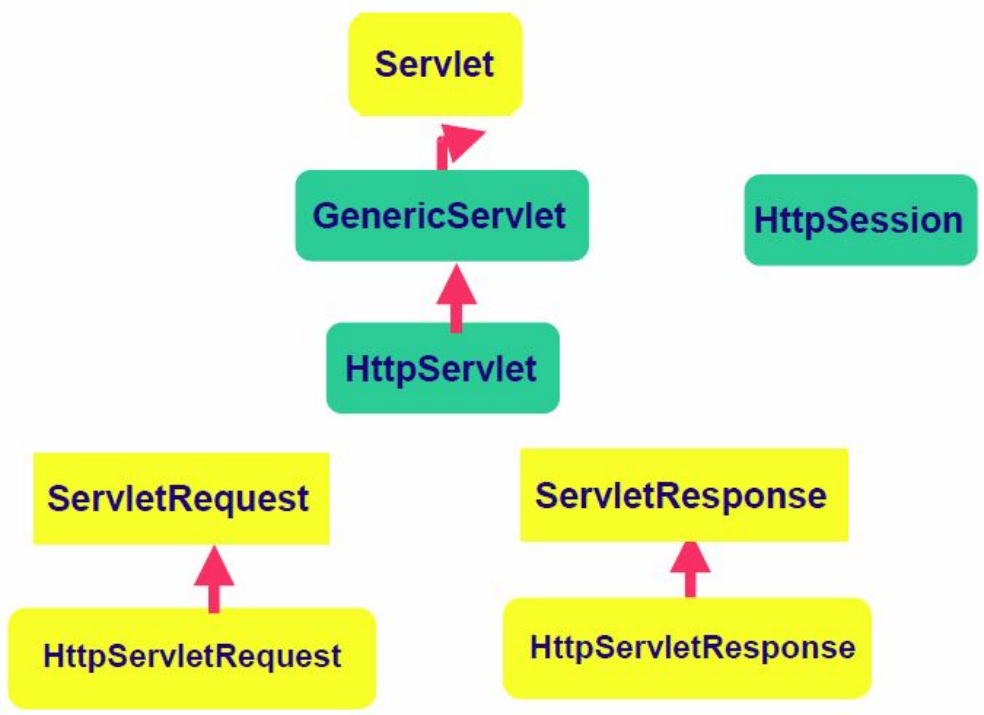
Servlet Interfaces & Classes



- **The *Serializable* interface provides the mechanism to implement Servlet session tracking, such as Servlet cookies.**

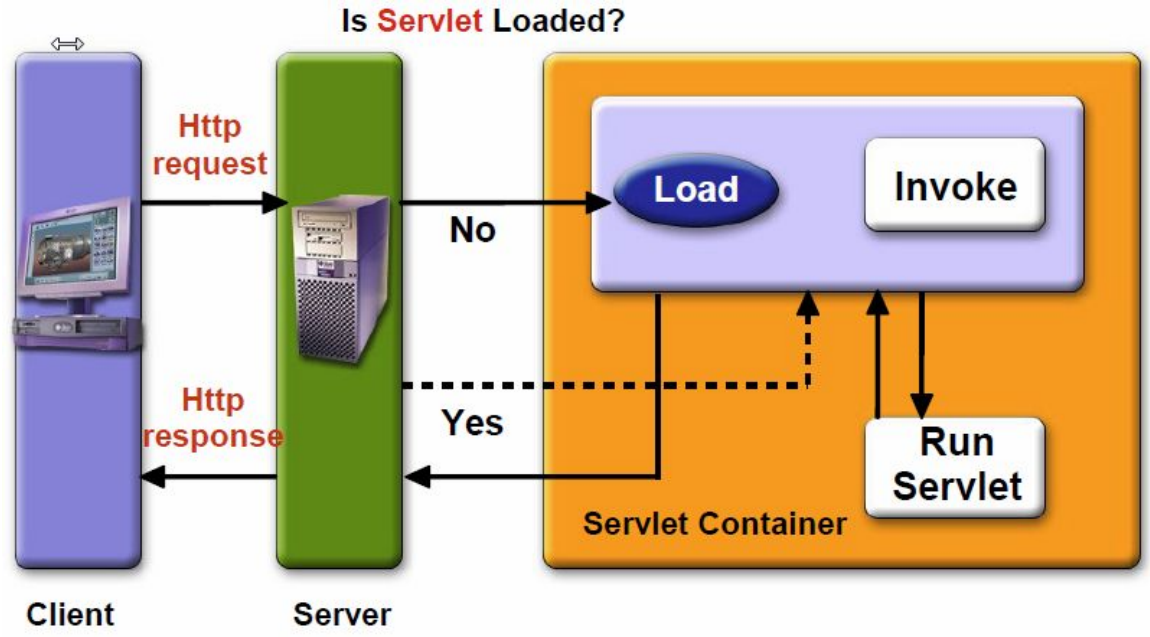
- **A subclass of *HttpServlet* must override at least one method, usually one of these: *doGet()* for HTTP GET requests and *doPost()* for HTTP POST requests.**

Servlet Interfaces & Classes

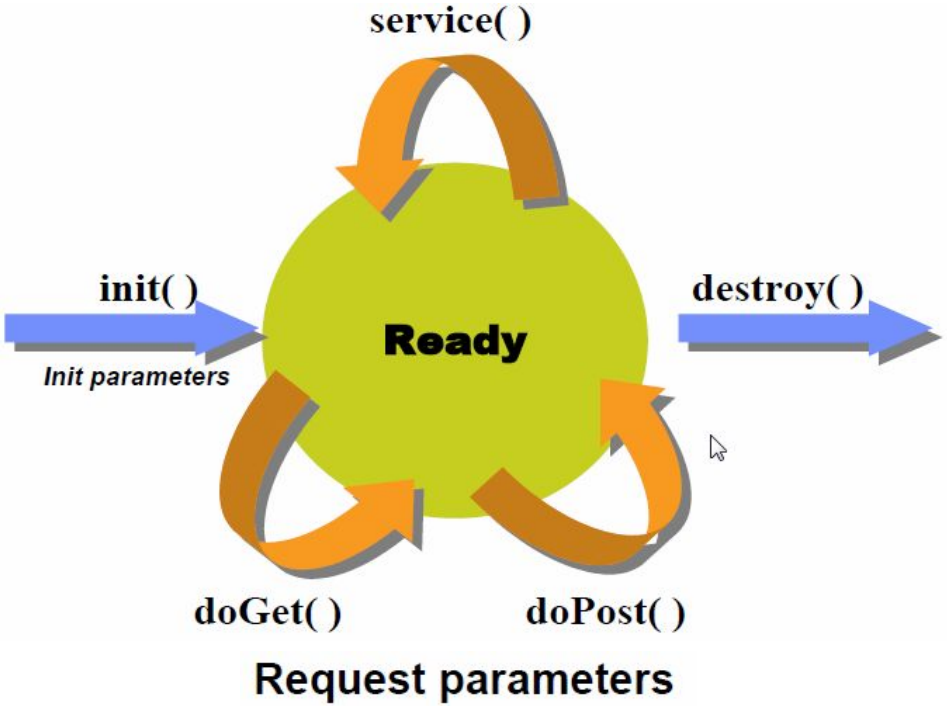


SERVLET LIFE-CYCLE

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

Example: init() method

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

// Perform any one-time operation for the servlet,
// like getting database connection object.

// Note: In this example, database connection object is assumed
// to be created via other means (via life cycle event mechanism)
// and saved in ServletContext object. This is to share a same
// database connection object among multiple servlets.

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

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;
  }
...
}
```

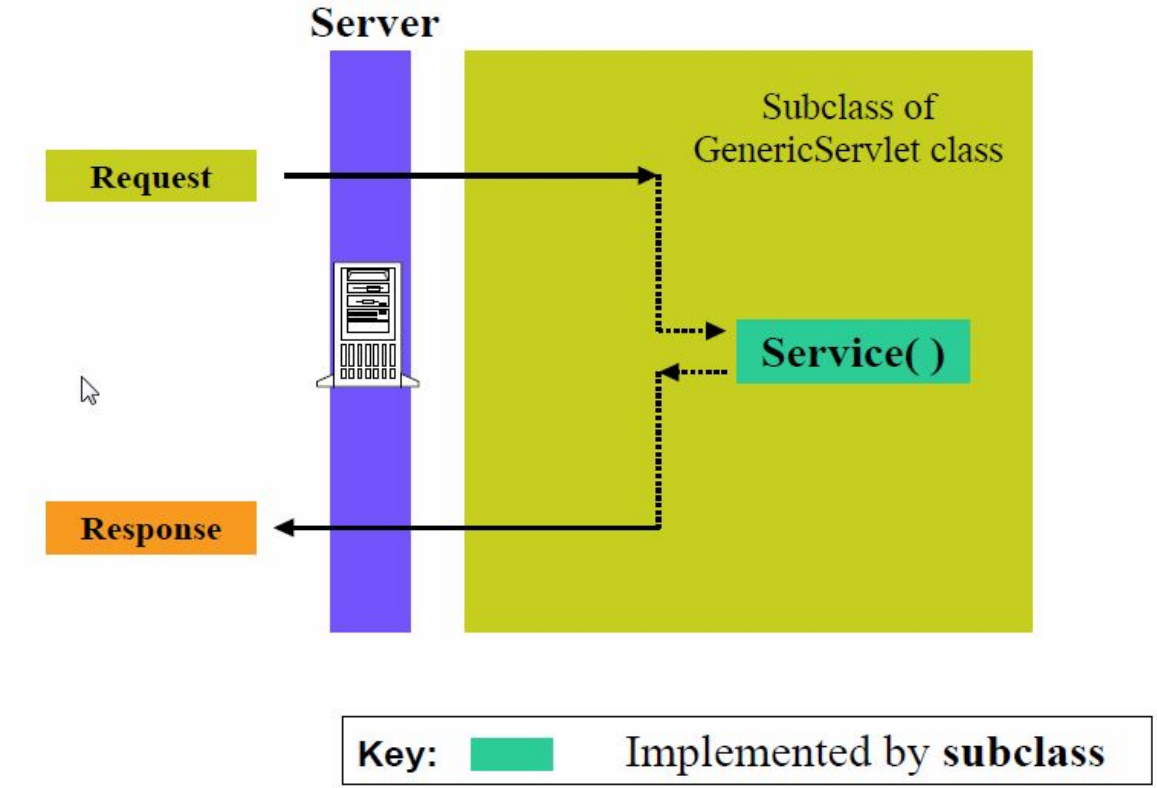
Servlet Life Cycle Methods

- 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

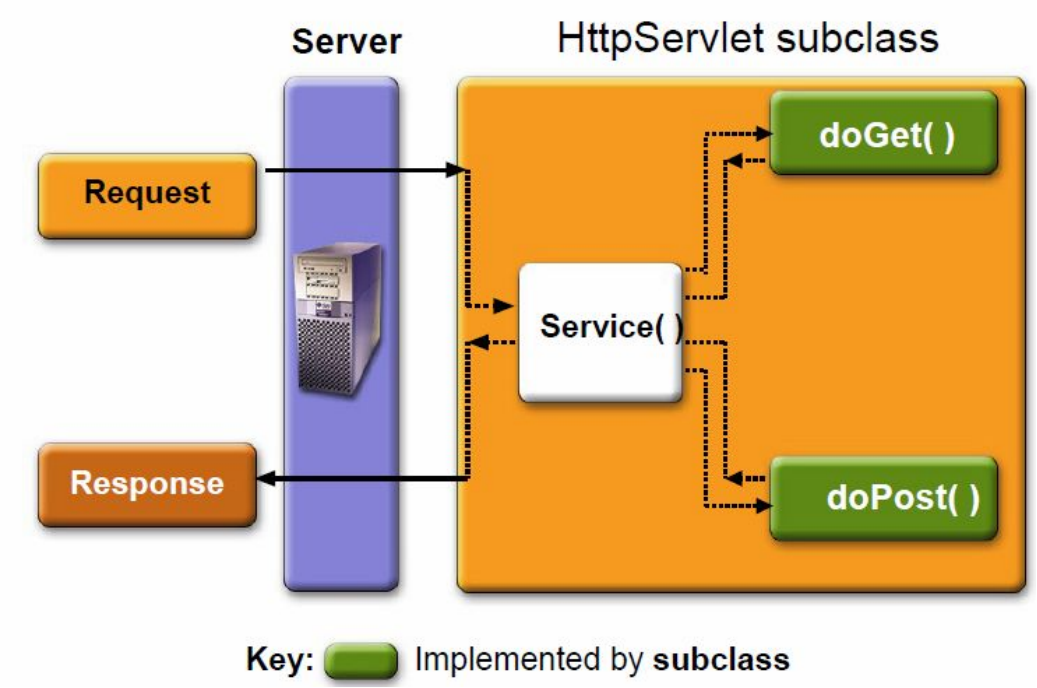
service() & doGet()/doPost()

- service() methods take generic requests and responses:
  - 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



doGet() and doPost() Methods



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 "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: 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>");
  }
}
```