Introduction to Servlets and Web Containers

Actions in Accord with All the Laws of Nature

Overview of Web Dynamics

- Interaction between a browser (client) and a web server:
  - Client requests a resource (file, picture, etc)
  - Server returns the resource, or declares it's unavailable

- Steps:
  1. User clicks a link or button in browser
  2. Browser formats request and sends to server
  3. Server interprets request and attempts to locate resource
  4. Server formats a response and sends to browser
  5. Browser renders response into a display for user

What Do Web Servers Serve?

- Without making use of some kind of helper application, a web server serves static content – text files, images, pdfs, videos, exactly as they are on the server machine.
- Responses cannot be customized based on input data passed in by the client or modified at the time they are delivered.

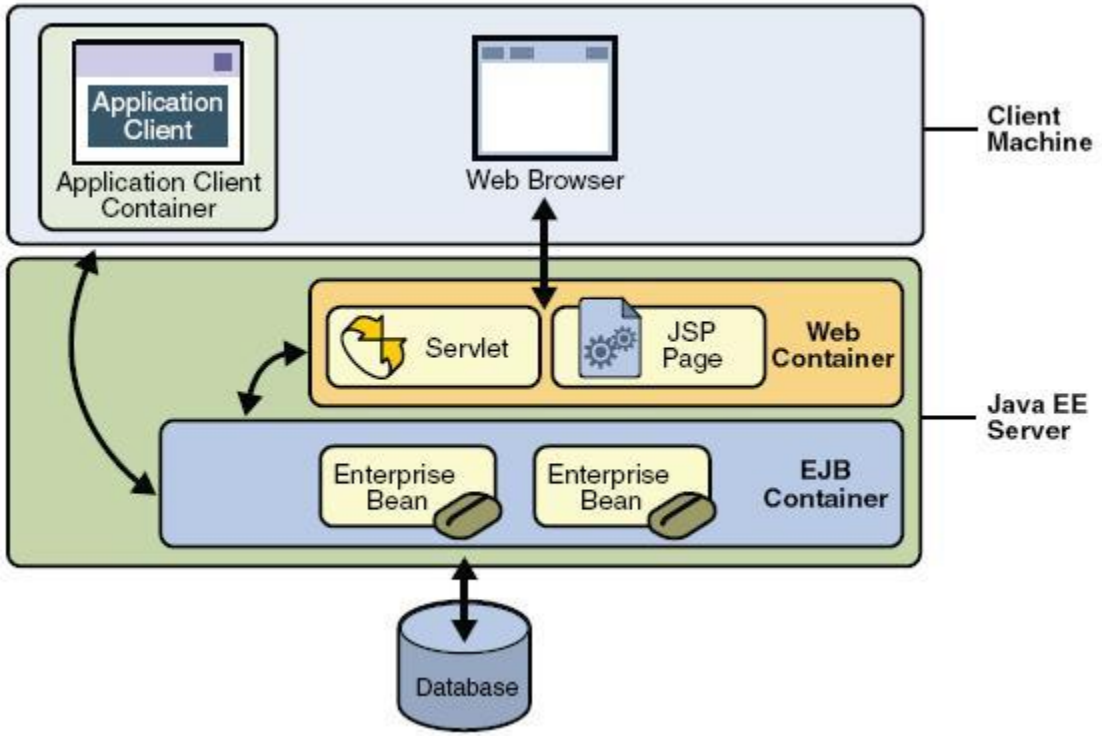
- A web server on its own can’t handle behavior like log in, online shopping, data lookups, and computations related to input data -- over the web -- require more than a web server.
- A web server on its own can’t save data to the server.

Servlets: Add Dynamism

- A Servlet is server-side java code (class) that can handle http requests and return dynamic content in http response.

- Servlets are managed by a servlet engine or container.
  - Each request that comes in results in the spawning of a new thread that runs a servlet (eliminating the cost of creating a new process every time).

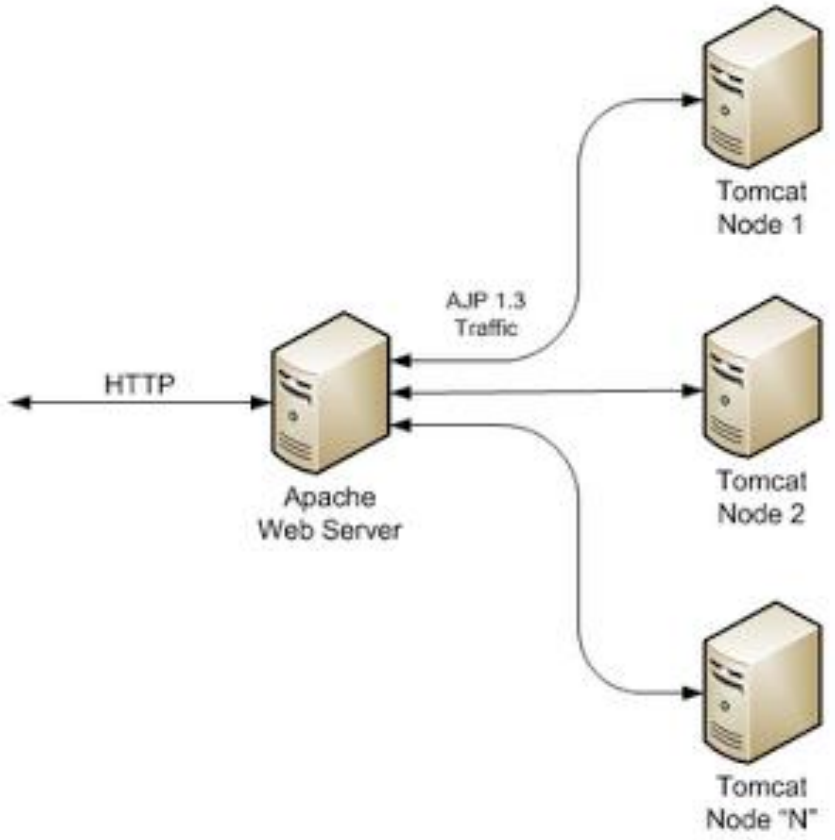
Web container and servlet architecture



A servlet is a Java class that extends the capabilities of servers that host applications access by means of a request-response programming model.

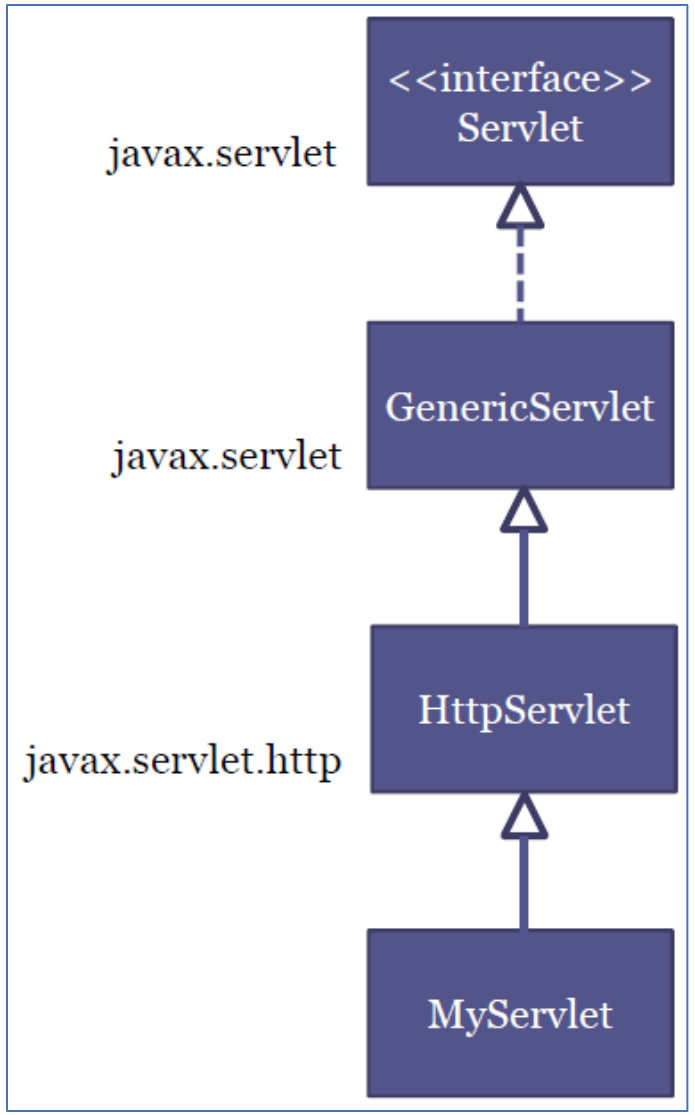
Web server vs web container

- Most commercial web applications use Apache web server
  - proven architecture and free license.
- Tomcat can act as simple servlet web container
  - for production environments it may lack features like load-balancing, redundancy, etc.
- Glassfish, WildFly are like Tomcat, but is JavaEE container



Servlet Hierarchy

- Servlet
  - service()
  - init()
  - destroy()
- GenericServlet
- HttpServlet
  - doGet()
  - doPost()
  - doPut()
  - doHead()
  - doDelete()
  - doTrace()



My Servlet

```
// MyServlet.java

public class MyServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
         PrintWriter out = resp.getWriter();
         out.print("Hello from my first servelt.");
    }
}
```

Servlet mapping using web.xml (DD)

```
<servlet>
    <servlet-name>hello-servlet</servlet-name>
    <servlet-class>wap.mum.edu.HelloServlet</servlet-
class>
</servlet>

<servlet-mapping>
    <servlet-name>hello-servlet</servlet-name>
    <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

- `servlet-name` is the internal name of the servlet
- `serlvet-class` is the Java name of the class
- `url-pattern` is the way the Servlet is specified in url bar or in the HTML form (public name)
  - `<form action="hello"  method="get">`

Mapping Requests

- **Exact match**

```
// must begin with /
<url-pattern>/courses/cs472</url-pattern>
<url-pattern>/courses/cs472.do</url-pattern>
```

- **Directory match**

```
//it works even when no Book directory exists (virtual mapping)
<url-pattern>/courses/*</url-pattern>
```

- **Extension match**

```
<url-pattern>*.do</url-pattern>
```

Annotated servlets

- Servlets can be declared and mapped using annotations instead of xml

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
    ...
}
```

XML vs Annotations

XML

- Centralized file to change the project
- Still need to use XML when annotations don't get the job done.

Annotations

- Annotations are in the same file which makes it easy to track and understand the purpose of the file
- Change should be done in all files

Note: XML configurations can be mixed with annotation-based configuration

Writing your first servlet

- Read the steps for *starting-with intellij*
- Note the following:
  - The index page (now it is called `index.jsp`)
  - What will happen when you click on the hyperlink?
  - When will `doGet` be called and what will it do?
  - When will `doPost` be called and what will it do?

Context `Init` parameters

- Context *initialization parameters* usually shortened to *init parameters*.
  - Can be put to any number of uses, from defining connection to database, to providing support email address.
  - You declare context init parameters using `<context-param>` tag with in the `web.xml` (no annotation alternative, you have to use DD)

```
<context-param>
    <param-name>support-email</param-name>
    <param-value>cstech.mum.edu</param-value>
</context-param>
```

```
ServletContext sc = this.getServletContext();
sc.getInitParameter("support-email");
```

Servlet `Init` Parameters

- You can also obtain inti parameter from the `ServletConfig` object.

```
ServletConfig sc = this.getServletConfig();
sc.getInitParameter("database");
```

```
<servlet>
    <servlet-name>...</servlet-name>
    <servlet-class>...</servlet-class>
    <init-param>
        <param-name>database</param-name>
        <param-value>dbCustomer</param-value>
    </init-param>
</servlet>
```

- Creates init parameter specific to this Servlet.
  - Can opt to use annotation instead.
    - drawback is, you need to recompile the application after every change.

Main Points

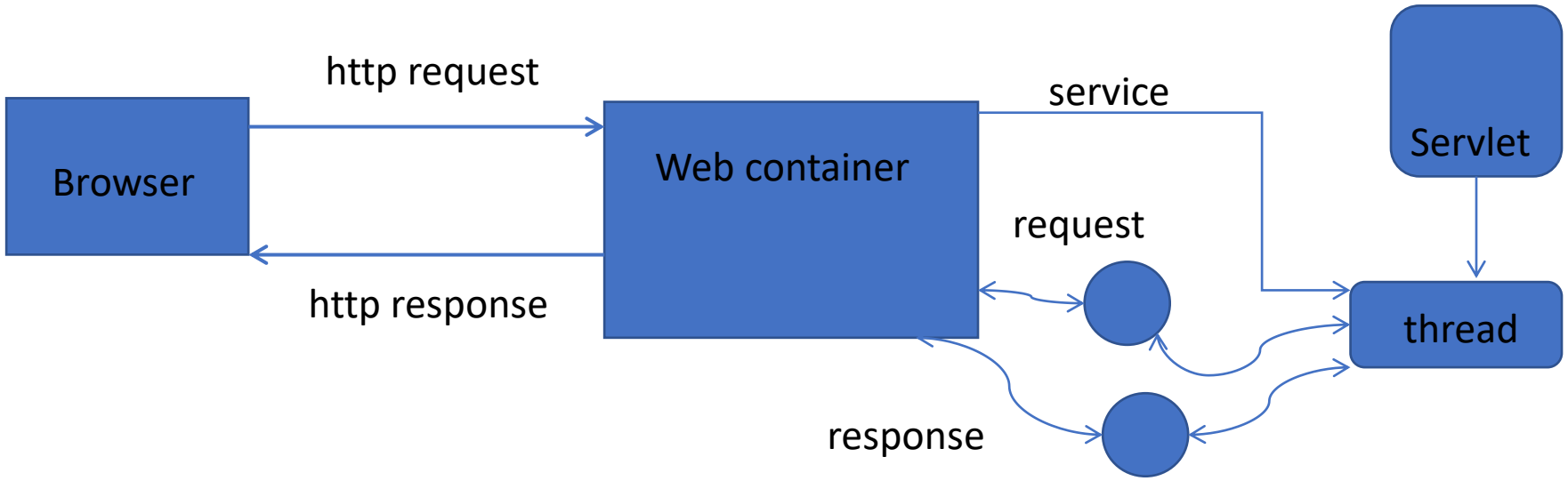
Every platform for web applications has a mechanism to dynamically generate web pages containing information from the server. On the Java platform Servlets are the Java classes that provide this dynamism.

Science of Consciousness: Pure consciousness is a field of infinite dynamism. We experience this as restful alertness during the practice of the TM Technique.

The Container

- Servers that support servlets have as a helper application, a servlet container.
- When a request comes to the web server, if the server sees the request is for a servlet, it passes the request data to the servlet container.
- The servlet container locates the servlet, creates request and response objects and passes them to the servlet, and returns to the web server the response stream that the servlet produces.
- The web server sends the response back to the client browser to be rendered.

How container handles an HTTP request



- Container receives new request for a servlet
- Creates `HttpServletRequest` and `HttpServletResponse` objects
- Obtains a new thread and calls `service` method on `HttpServlet` object in thread
- When thread completes, converts response object into HTTP response message

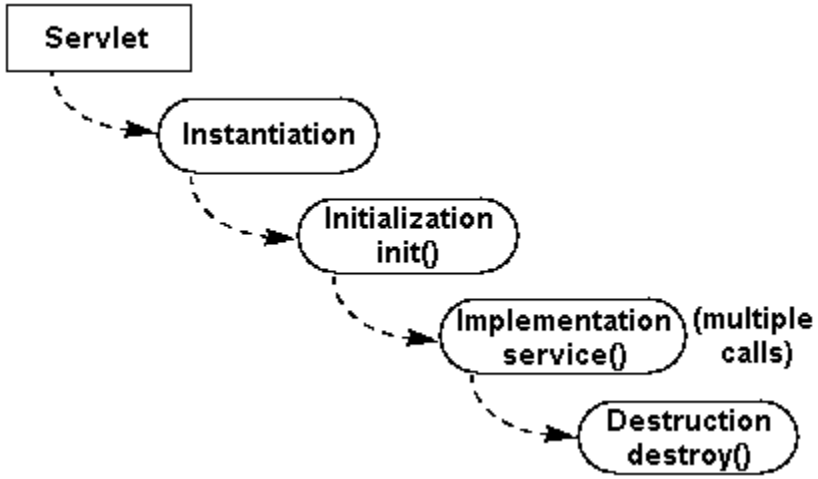
Containers provide fundamental support

- network communications
  - communicating with web server
- lifecycle management
  - no "main" method in a servlet, …
- concurrency
- state management
  - request, session and context attributes
- security
- Support for JSP (JSF, JPA, JTA, EJB, …)

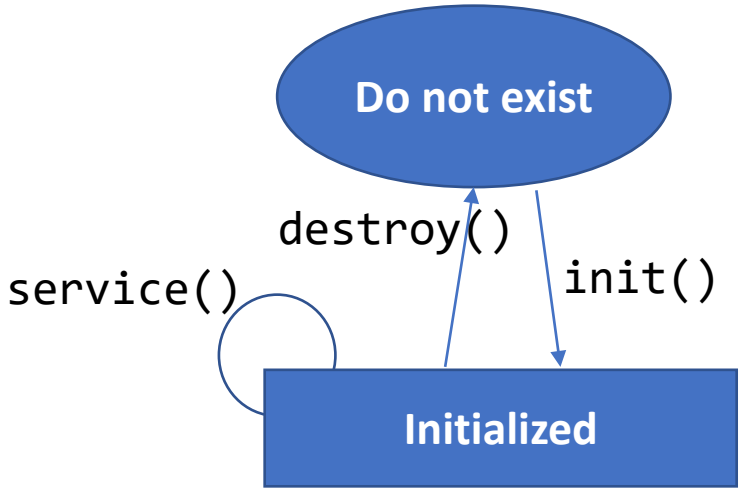
The `service()` method

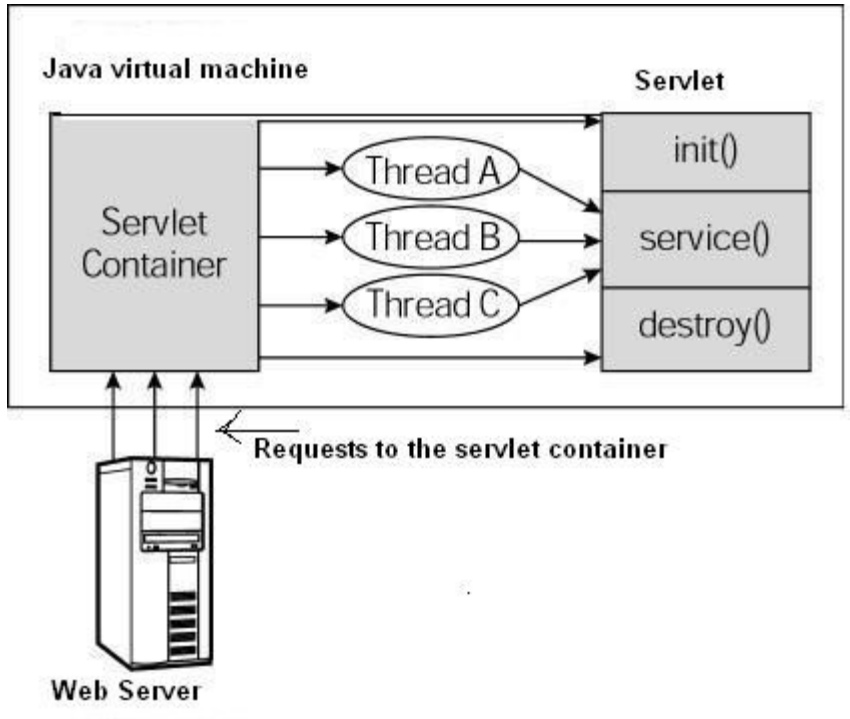
- The servlet container (Tomcat) calls the `service` method to handle requests coming from the client (browsers) and to write the formatted response back to the client.
- Each time the server receives a request for a servlet, the server starts a new thread and calls `service`.
  - The `service` method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls `doGet`, `doPost`, `doPut`, `doDelete`, etc. methods as appropriate.
- Threads share the same instance (and instance variables) of the servlet class.
  - Every method call (e.g., service) in a thread will have its own space on the call stack and own local variables.

Servlet life cycle



. Load
. Create
. Init
. Service
. Destroy





- Single instance of servlet
- new thread created for every request
  - Then `service` called on the thread
- All threads **share instance variables**
- Each thread has **own** stack for **local variables**

Using the Initializer and Destroyer

- `init` and `destroy` methods do nothing by default.
  - But you can override then to perform some actions:

```
@Override
public void init() throws ServletException {
    System.out.println("Servlet " + this.getServletName() + " has started.");
}

@Override
public void destroy() {
    System.out.println("Servlet " + this.getServletName() + " has stopped.");
}
```

Using the Initializer and Destroyer

- `init` is called after the Servlet is constructed but before it can respond to the first request.
  - Unlike when the constructor is called, when `init` is called all the properties have been set to the Servlet, giving you acess to the `ServletConfig` and `javax.servlet.ServletContext` objects.
  - You may use this method to read a properties file or connect to database etc.

- Likewise, `destroy` is called immediately after the Servlet can no longer accept the request.
  - This typically happens when the web application is stopped or undeployed or when the container shuts down.
  - Because it is called immediately upon undeployment or shutdown, you don't have to wait for garbage collector before cleaning of resources.

When does servlet get instantiated?

- Usually, the servlet is instantiated and init method is called when the first request arrives for the Servlet after the web application starts.
  - Normally, this is sufficient for most uses.
- However, if the `init` method does many things, Servlet startup might become a time-intensive process, and this could make the first request to that Servlet take several seconds or even several minutes! Obviously, this is not desirable.
  - A simple tweak to the servlet configuration can make the servlet immediately when the web application starts:

```
<servlet>
    <servlet-name>...</servlet-name>
    <servlet-class>...</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

Service Parameters

- `HttpServletRequest` implements `ServletRequest`.
- `HttpServletResponse` implements `ServletResponse`.

- Managed by the container, passed to the servlet.
- Servlet sends them back to container to help it send the response back to the browser.

HttpServletRequest

- Using the request, the service method can tell if the http request was a GET or POST and delegate to the appropriate method in the servlet class.
- Get header information
  - `request.getHeader(name);`
- Get Session & Cookie related to the request.
- Get User information
- Get Path and url

Getting Request Parameters

- Perhaps the most important capability of `HttpServletRequest`, is to retrieve request parameters passed by the client.
- Request parameters comes in two different forms:
  - via query parameters
  - or as post variables or form variables
- The Servlet API does not differentiate between two type of parameters.
  - A call to any of the parameter-related methods on a request object returns parameter whether they were delivered as query parameters or post variables.

Getting Request Parameters

- The `getParameter` method returns a single value for a parameter.
  - If the parameter has multiple values, `getParameter` returns the first value
  - `getParameterValues` returns an array of values for a parameter.
    - If the parameter has only one value, this method returns an array with one element in it.

- HTML

```
<input name="userName" type="text" />
```

- Servlet

```
String input = request.getParameter("userName");
```

```
request.getParameterValues("checkbox_name");
```

Using `HttpServletResponse`

- `HttpServletResponse` interface extends `ServletResponse`
- You use the response object to do this such as:
  - set response headers

```
response.setContentType("text/html");
response.setCharacterEncoding("UTF-8");
```

  - write to the response body

```
PrintWriter out = response.getWriter();
out.print("<html><head><title>Test</title></head><body>");
```

  - redirect the request
  - set the HTTP status code
  - send cookies back to the client

Multithreading and Thread Safety

- Web applications are, by nature, multithreaded applications.
  - At any given time, zero, one, or a thousand of people may be using your web application simultaneously.
- The most typical complication when coding for a multithreaded application is the access of shared resources.
  - static and instance variables in a Servlet, for example, could be accessed by multiple thread simultaneously.
  - It is important to synchronize access to these shared resources to keep their contents from being corrupt and possibly causing errors in your application.
    - Use synchronized block if necessary.

Main Point 3

Web containers manage the life cycle of servlets and insure that everything happens as required with minimal involvement of web developers. ***Science of Consciousness***: The unified field manages the entire universe, and by experiencing this field of awareness on a regular basis we spontaneously act more and more in accord with all the laws of nature.

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

Web Containers: Actions in Accord with All the Laws of Nature

1. Developers override the doGet or doPost methods of servlets to implement the request-response functionality of the web application.

2. The web container is responsible for calling the service methods as well as managing the lifecycle of the servlet and exchanging all information over the network.

3.Transcendental consciousness is the experience of the home of all the laws of nature. Having this experience structures one's awareness to be in accord with all the laws of nature.

4.Impulses within the Transcendental Field: Servlets represent specific impulses of intelligence that are supported by the general-purpose services of the web container. In a similar manner, thoughts connected with the transcendental field are supported by all the laws of nature.

5.Wholeness moving within itself: In unity consciousness, thoughts and actions arise from this level of thought, and daily life is lived in terms of this experience of wholeness and integration. This is like the effects of integration and correctness that are produced in web applications due to the underlying wholeness and integration of the web container.