ICON OF JAVA

ADV. JAVA
Servlets

4. Servlets lifecycle

Mr. Nagoor Babu M.Tech

[ICON of JAVA]
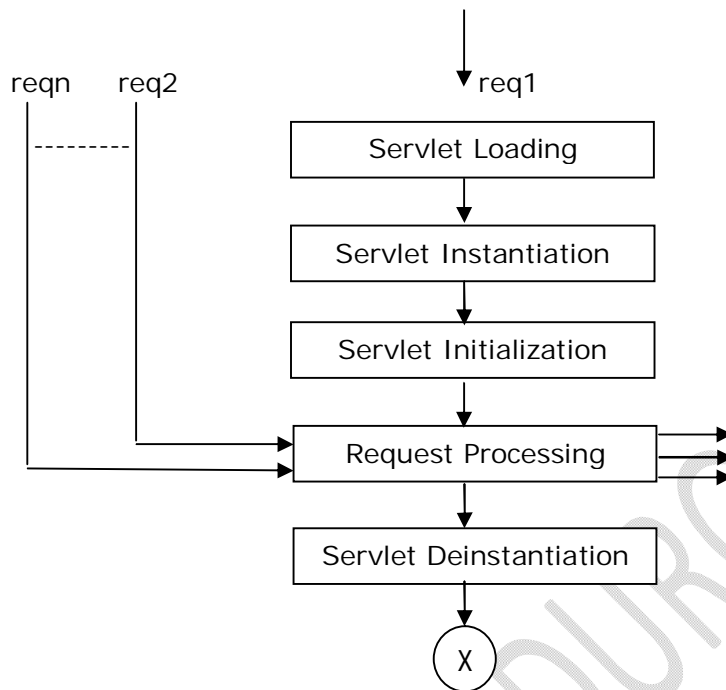**(Sun certified & Realtime Expert)**
Ex. HCL Employee
Trained Lakhs of Students
for last 14 years across INDIA

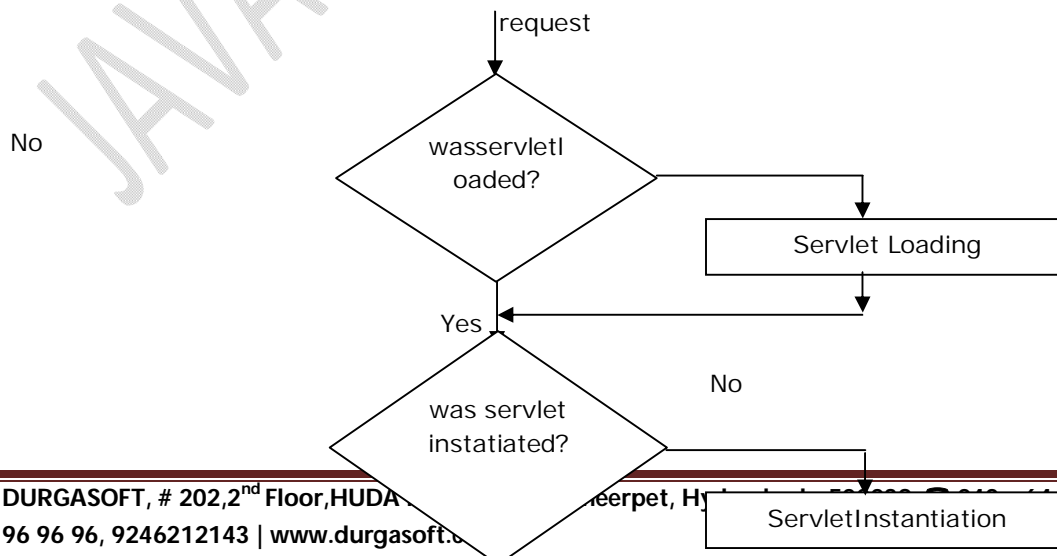## India's No.1 Software Training Institute
DURGASOFT
www.durgasoft.com Ph: 9246212143 ,8096969696

Servlet Life Cycle

```
                              req1
              reqn    req2     ↓
                    ┌──────────────────────┐
              ┌ ─ ─ ┤   Servlet Loading    │
              │     └──────────┬───────────┘
              │     ┌──────────▼───────────┐
              │     │ Servlet Instantiation│
              │     └──────────┬───────────┘
              │     ┌──────────▼───────────┐
              │     │ Servlet Initialization│
              │     └──────────┬───────────┘
              │     ┌──────────▼───────────┐  →→
              └────►│  Request Processing   │  →→
                    └──────────┬───────────┘  →
                    ┌──────────▼───────────┐
                    │Servlet Deinstantiation│
                    └──────────┬───────────┘
                              (X)
```
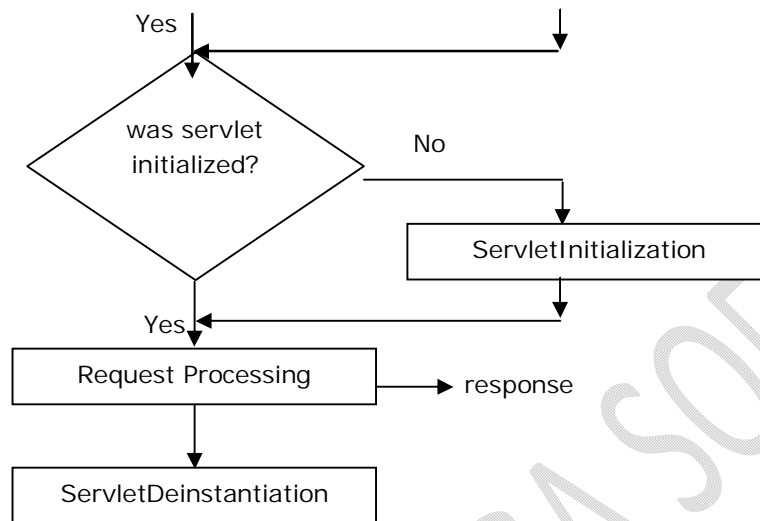
From the above representation when we send multiple number of requests to a particular servlet at a time then container will perform Servlet Loading, Instantiation and Initialization only for first request, container will bring all the later request directly to Request processing phase by skipping Servlet Loading, Instantiation and Initialization phases.

   If we send multiple number of requests to a particular servlet then container will perform the following work for each and every requests to execute life-cycle stages.

```
                        request
                          │
                          ▼
                    ╱───────────╲
   No              ╱  wasservletl  ╲         ┌──────────────────┐
                  ╲    oaded?      ╱ ───────►│  Servlet Loading │
                   ╲─────────────╱           └────────┬─────────┘
                        │ ◄───────────────────────────┘
                   Yes  │
                    ╱───────────╲
                   ╱             ╲        No
                  ╲  was servlet  ╱ ───────────┐
                   ╲ instatiated? ╱            ▼
                    ╲───────────╱      ┌──────────────────┐
                                       │ServletInstantiation│
                                       └──────────────────┘
```

From the above representations all the servlets and Jsp's are able to allow multiple number of requests at a time i.e. multiple number of threads at a time and all the servlets are able to process multiple number of threads at a time without providing data inconsistency. Therefore all the servlets and Jsp's are by default Thread safe.

Note: If any resource is able to allow and process multiple number of threads at a time without having side effects then that resource is called as **Thread Safe Resource.**

In servlet applications, as per the application requirement if we want to allow only one request at a time to a particular servlet then Servlet API has provided javax.servlet.SingleThreadModel marker interface.

public class MyServlet extends HttpServlet implements SingleThreadModel

{ ----------------- }

Where SingleThreadModel interface is a deprecated interface provided by Servlet API at the initial versions of servlet, which was not supported by the latest versions of servers.

Still, if we want to achieve our requirement about to allow only one request at a time when we have to use synchronization in the respective servlets.

Among the synchronization it is suggestible to use synchronized blocks over synchronized methods in order to improve performance.

```
public class MyServlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res)throws SE, IOE

    { ---------

      ---------

      synchronized(this)

      { ---------

        --------- }

    }

}
```

Note : In web applications, always it is suggestible to avoid to use SingleThreadModel and Synchronization because they will decrease the performance of web application.

Q: As part of servlet design if we access destroy() method in init() method then what will be the response from servlet?

Ans: As part of servlet design if we access destroy() method in init() then container will not perform servlet deinstantiation as part of servlet initialization.

   Container is a software which includes number of modules to perform Servlet life cycle. When we send a request to the respective servlet from a client then container will start Servlet life cycle by executing its internal modules.

   While performing servlet initialization, container will access init() method, init() method is not responsible to perform servlet initialization completely.

   Similarly container will perform servlet deinstantiation by executing its internal modules. As part of this, container will access destroy() method.

   Due to the above reasons eventhough we access destroy() method in init() method then we are able to access our provided destroy() method just like a normal method, it will not perform servlet deinsatntiation.

Q: If we send GET request from user form but if we override doPost(_,_) in the corresponding servlet what will be response from servlet?

Ans: The main convention of HttpServlet is, if we specify xxx request type t

hen container will execute doXxx(_,_) method i.e. we will override doXxx(_,_) method.

With the above convention, if we specify GET request at user form then we must override doGet(_,_) method.

   In the above content, if we override do Post(_,_) method for GET request then container will execute predefined doGet(_,_) method provided by HttpServlet as for the predefined implementation of service(_,_) method.

   In HttpServlet, doGet(_,_) method was implemented in such a way to send an error message with the following error description.

HTTP Status 405-HTTP method GET is not supported by this URL

Note: In HttpServlet, by default all doXxx(_,_) methods are implemented with the above convention only.

Note: In case of Tomcat server, if we specify any request type except GET and POST at user form then container will treat that request type has default request type and it will execute doGet(_,_) method.

   This type of implementation may not be available with all remaining servers. With the above implementation, Tomcat server has provided support for only GET and POST request types.