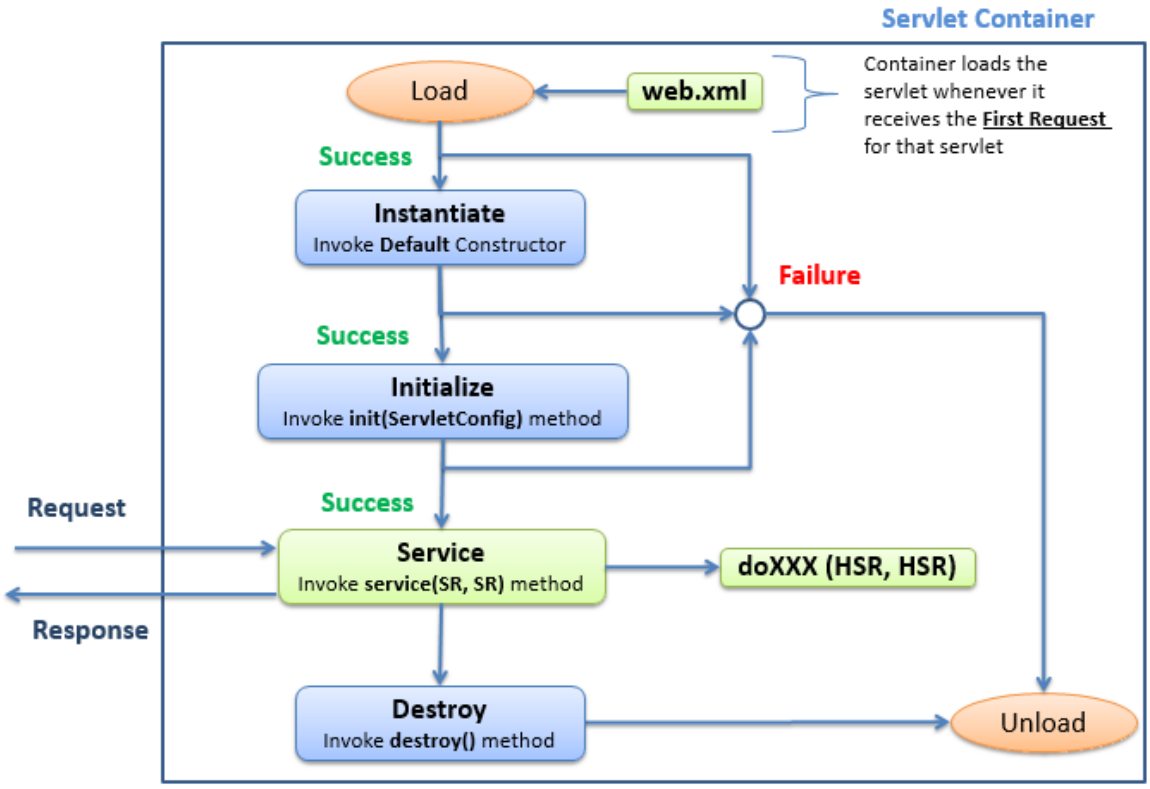
Servlet Life Cycle

Life-cycle of a Servlet is controlled by Servlet Container & it has following phases

1. Instantiation Phase
2. Initialization Phase
3. Service Phase
4. Destruction Phase



Instantiation Phase:

- Whenever request comes to a container, by looking at the URL & referring web.xml container tries to find the Servlet name.
- If NO Servlet found, then it returns "404 Error Response".
- If Servlet found, then Container creates an instance of the Servlet by invoking "Public Default Constructor ONLY".

Initialization Phase:

Version 1:

```
public void init(ServletConfig config) throws ServletException
{
        super.init(config);
        //Initialization Code Goes Here
}
```

Version 2:

```
public void init() throws ServletException
{
        //Initialization Code Goes Here
}
```

- After Successfully creating an instance, Container automatically invokes "init(ServletConfig)" method.
- init(SC) method gives us a chance to initialize the Servlet before handling the requests. Like,
  - Opening a Text File or
  - Reading the data from a Property File, etc.
- This method is called ONLY ONCE in Servlet Life-cycle.
- We may/may-not override this method. If we don't override then default implementation present in GenericServlet is invoked.
- The first line of the Version 1 init(SC) method should be "super.init(config)".
- During initialization, Servlet has access to "javax.servlet.ServletContext" object .
- Once Instantiation & Initialization is successful, container caches the Servlet Instance.

Can we use Constructor for Initialization ?

- We can make use of Constructor for initialization "ONLY if initialization code is independent of ServletContext Object
- If initialization code is dependent on ServletContext Object then "we left with no choice" of using init method for initialization purpose.
- Also in case of constructor, we must make use of "Public Default Constructor"
- Hence we generally make use of init method for initialization purpose (dependent / independent of ServletContext / ServletConfig Objects) without touching constructor

Service Phase

public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException

- After Instantiation & Initialization, Container creates Request & Response Objects, invokes service(SR, SR) method by passing these objects.
- This method is called "for every request" i.e. one/more times in Servlet Life-cycle .
- Inside service(SR, SR) method we write any Java Code which needs to be executed for every request.
- If a Servlet is a sub-class of GenericServlet then we MUST override this method
- If a Servlet is a sub-class of HttpServlet then we SHOULD NOT override this method & our job is to override one/more doXXX() method.

Destruction Phase

```
public void destroy() {

    //Clean-up Code Goes Here

}
```

- Whenever container wants to remove the cached Servlet Instance from it's memory then it invoke destroy() method "before removing" destroy() method gives us a chance to perform any clean-up activity such as Closing a File etc.,
- This method is called ONLY ONCE in Servlet Life-cycle.
- We may/may-not override this method. If we don't override then default implementation present in GenericServlet is invoked.

Servlet Lifecycle Phases	Servlet.java (Interface)	GenericServlet.java (Abstract Class)	HttpServlet.java (Abstract Class)

1. Instantiation		init (SC) {
		----- 	
		this.init();
		}
			Inherited

2. Initialization	public void init (ServletConfig) throws ServletException	init () { } Empty Method	
3. Service	public void service (ServletRequest, ServletResponse) throws ServletException, IOException	Abstract Method	service (SR, SR) ↓ service (HSR, HSR) ↓ One of 7 doXXX (HSR, HSR)
4. Destruction	public void destroy ()	destroy () { } Empty Method	Inherited

NOTE:-

NO Matter how we create a Servlet, Container **ALWAYS** invokes below Life-cycle Methods on that Servlet

1. **Public Default Constructor**
2. **void init(ServletConfig)**
3. **void service(ServletRequest, ServletResponse)**
4. **void destroy()**

Important Points:

- Any Class which extends any one of the below class is called as a "Servlet"
  - **javax.servlet.http.HttpServlet**
  - **javax.servlet.GenericServlet**
- In-other-words Servlet MUST be an Object of type "**javax.servlet.Servlet**" interface Hence, there are 3 ways of creating the Servlet
  1. By extending **javax.servlet.http.HttpServlet**
  2. By extending **javax.servlet.GenericServlet**
  3. By implementing **javax.servlet.Servlet Interface**
- Servlet interface has below 5 Method

  - Life Cycle Methods:
  1. void init(ServletConfig config) throws ServletException
  2. void service(ServletRequest req, ServletResponse resp) throws ServletException, IOException
  3. void destroy()

  - Other Methods:

  4. ServletConfig getServletConfig()
  5. String getServletInfo()
- If we implements Servlet Interface then that Servlet will become "Protocol-Independent Servlet".
- If a class
  - extends either HttpServlet or GenericServlet OR
  - implemets Servlet interface
  - & a "subclass of that class is also be called as Servlet"
- Servlets (for which we configure a URL in web.xml) must be a "concrete class" otherwise they fail at runtime i.e. during the "Instantiation Phase"
- Servlets can have
  1. class level Variables (Static / Non-Static)
  2. Block of Code (Static / Non-Static)
  3. Innner Classes
  4. It's own Methods (Static / Non-Static but non-abstract)

Note:

- If we declare abstract method(s) in servlet then we are forced to declare that servlet as "Abstract Class" & we MUST sub-class it Otherwise, it fails at runtime during Instantiation.
- We can have "main() method" in the servlet but its of "no use". Servlets are directly under the control of Container & Container will not execute main method in any phases of Servlet Lifecycle.
- Servlets MUST have public default Constructor OR combination of any other constructor along with public default constructor

- Servlet API is protocol independent in nature, but are most often used with HTTP & HTTPS protocols
- There is only one instance exist for any servlet. i.e. Servlets are "Singleton in nature"
- At any point of time there will be multiple threads acting on servlet instance
- Hence by default servlets are Multi Threaded in Nature. In other words Dynamic Web Applications are "Multi Threaded environment".

`