LifeCycle, Request, Response and Web Container

https://www.youtube.com/watch?v=p1c_Qgp8Px0

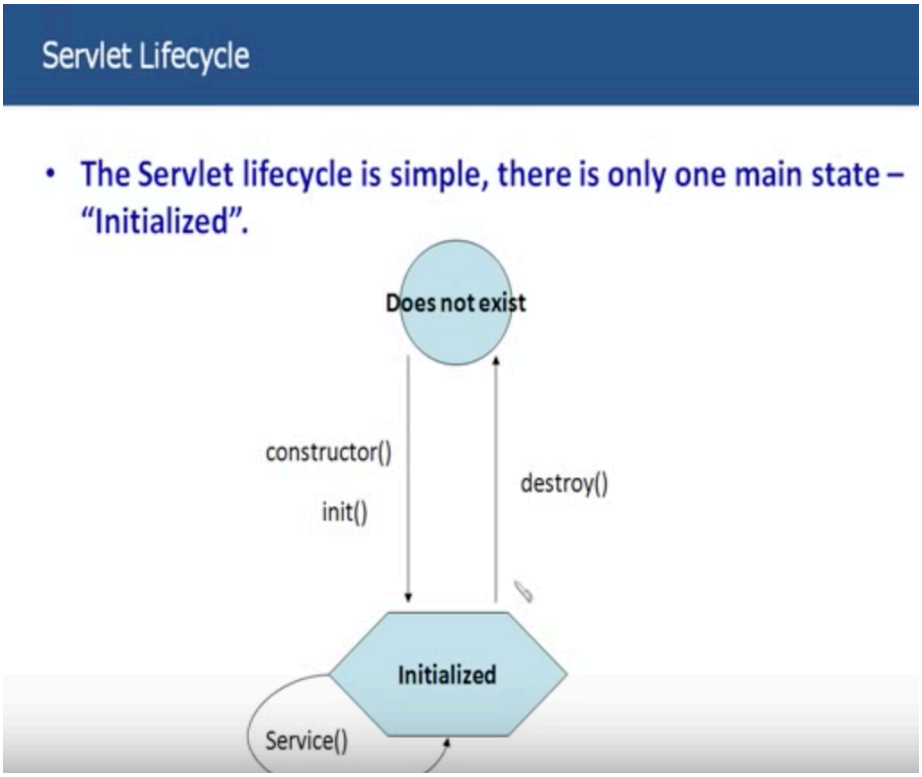
So far, we have discussed

- what a Servlet is
- why do we use a Servlet?
- How the communication takes place in between client and Servlet?

Now we are going to have a look at Servlet LifeCycle.

We are going to see:

- What are the different Lifecycle methods of Servlet are?
- What the different state of Servlet are?
- When are these methods called?



Basically, there is only state for a Servlet. That is **Initialized** state.
Does not exist state is merely void state. State in not real in terms.

So let's discuss how the state move from Does not exist state to Initialized state and how the Servlet state is changed from Initialized to Does not exist.

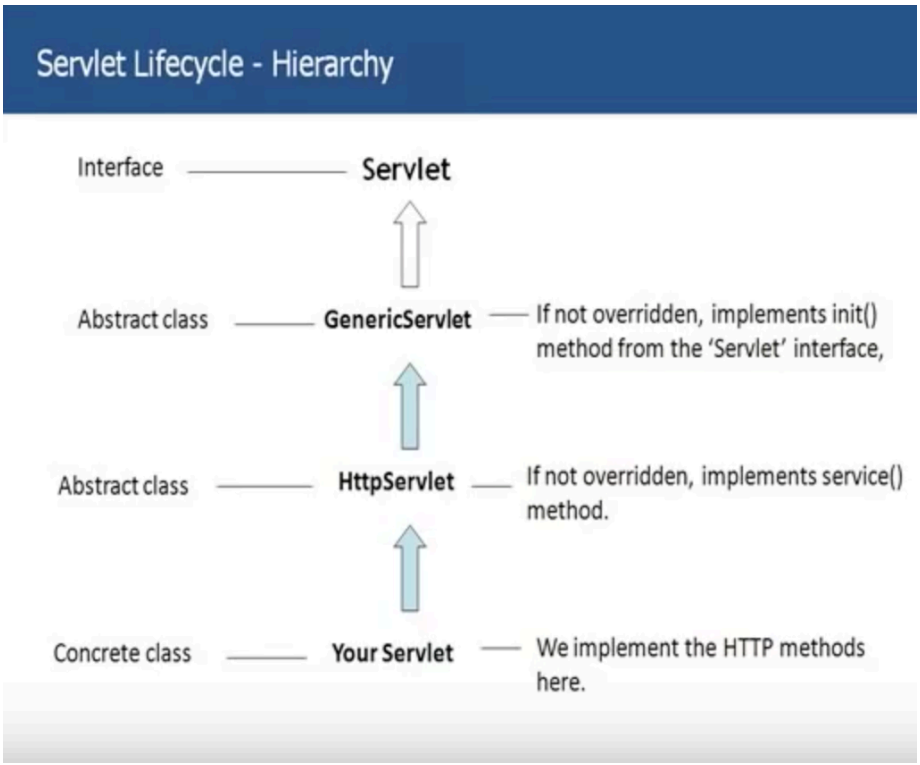
When the first request for the servlet comes in; constructor() and init() methods are called on the Servlet. Only in life time of a Servlet constructor() and init() methods are called. And once the init() method is called Servlet moves from Does not exist state to Initialized state and ready to be used by other requests, I mean by other clients.

So, for every subsequent request that comes in, service() method is called and Servlet still stays in initialized state. When the destroy method is called, the Servlet again from the initialized state to the Does not exist state. And all these methods are called by Web Container or Servlet Engine.

Will see in later slides:

- when are these methods called?
- how are these methods called?
- what's the use of it?

Hierarchy:



This is the Servlet hierarchy, so where your Servlet behavior comes from you can see the sketch here.

- At the top of this hierarchy, you have the Servlet Interface.
- GenericServlet implements this Servlet interface actually an abstract class. If you do not override init(), if you don't have any initialization code in your servlet, you don't need to override the init(). When you do not override init() method takes default implementation from the GenericServlet.
- Now HttpServlet extends GenericServlet and this again is an abstract class, and here if you do not override service(), default implementation in HttpServlet is taken.
- And in the end, your servlet. This is the Servlet which you code in and the actual class should extend the HttpServlet and this is the concrete class. And this is the place where you implement your Http methods i.e., either doGet() or doPost().

Let's now discuss different methods of a Servlet and the details of the method like when are they called and what is used for and all those different things.

Servlet Lifecycle – 3 big moments

	When is it called	What it's for	Do you override it
init()	The container calls the init() before the servlet can service any client requests.	To initialize your servlet before handling any client requests.	*Possibly*
service()	When a new request for that servlet comes in.	To determine which HTTP method should be called.	*No. Very unlikely*
doGet() or doPost()	The service() method invokes it based on the HTTP method from the request.	To handle the business logic.	*Always*

init():

When is it called?

- The container calls the init() before the servlet can service any client requests.

What it's for?

- To initialize your servlet before handling any client requests. Initializing can be initializing your data source to connect to your database or any other such initializations you want to perform.
- What is Initialization? It's a one-time process. You don't that every time for every client request that this particular thing to happen. So you put this kind of code in initialization. So, it happens only once in lifetime of a Servlet.

- Now a question might have come up, we have constructor for a Servlet? Generally, when you talk about Java, we put all out initialization stuff in the constructor of the class. So, why do we have a separate method for initialization? Because, init() method has access to 2 special objects for ServletContext and ServletConfig. And generally you use those objects in your init() during initialization.

Do you override it?

- Possibly

Service():

When is it called?

- It is called for every request that comes in for the Servlet. If there is a YahooLoginServlet, every time a person makes a request to YahooLoginServlet the service() of this YahooLoginServlet is called.

What it's used for?

- This is used to determine which http method should be called.

Do you override it?

- No. Very unlikely.

doGet() or doPost():

These are the 2 methods where you put all your business logic. If type of request is GET, you override the doGet(). If the type of request is POST, you override the doPost().

When it is called?

- The service() invokes it based on HTTP method from the request.

What it's used for?

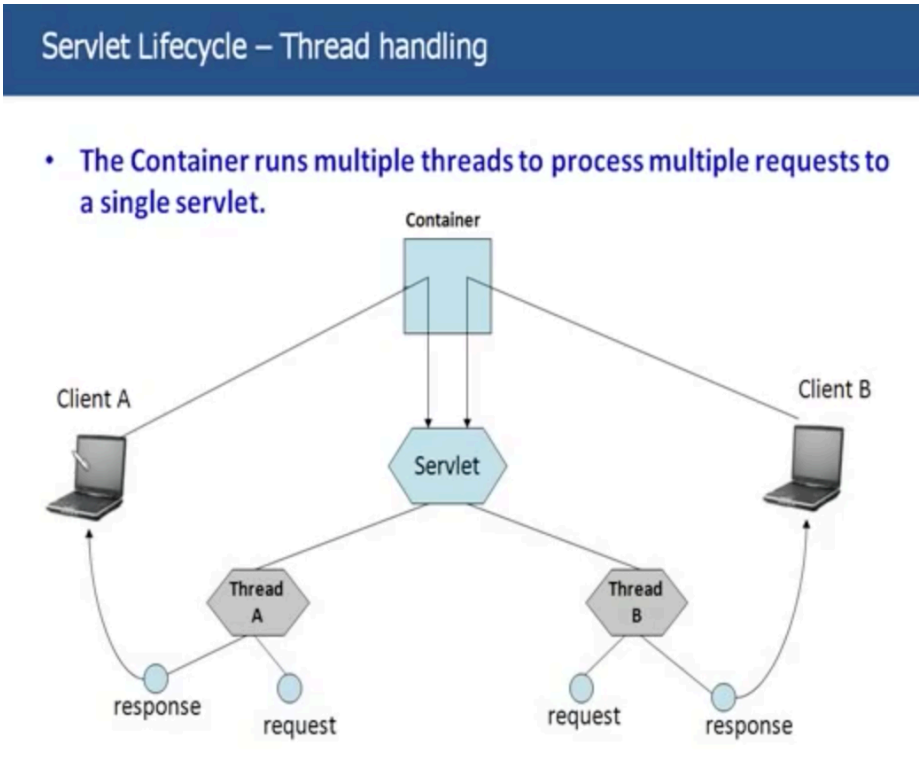
- To handle business logic.

Do you override it?

- Always

Note: You also have destroy(), which is called by the Servlet when the time out or the Servlet is no longer required. The container calls the destroy().

Thread handling:



We actually talk that; the web container provides multi threading support for the Servlets. So, lets have a detail look how that is implemented.

Let's take an example where you are login to Yahoomail.com and you sending an email, username and password, then YahooMailServlet will actually give you back your Inbox with all your messages.

Let's say client A logs in to yahoomail with username as abc. What the container does, the container sponsor a Thread A for this client A pass in request as abc to Thread A.

At the same time client B logs in to yahoomail with username as xyz. What container does, the container will sponsor another Thread called Thread B for this client and pass the request with xyz to the Thread B.
So, Thread A process will process the request abc, create a response for that and sends back the response to client A. Similar way, Thread B will process the request xyz, create a response and send back the response to client B.
These are 2 separate threads respond, but the request and response objects are different. They do not share the same request and response object.
So, Thread A has its own request object and Thread B has its own response object.
Basically, this is how multi threading works for a Servlet.

Request and Response - GET v/s POST:

We have been discussing about GET and POST methods. We have seen that when type of request is GET from client, we use doGet(), type of request from client is POST, we use doPost().

Big Question is when do you go for a GET and when do you go for a POST?

I am just going to talk about differences between GET and POST, which will ultimately let us understand when we use a GET and when we have to go for a POST.

Request and Response – GET v/s POST

- **The HTTP request method determines whether doGet() or doPost() runs.**

	GET (doGet())	POST (doPost())
HTTP Request	The request contains only the request line and HTTP header.	Along with request line and header it also contains HTTP body.
Parameter passing	The form elements are passed to the server by appending at the end of the URL.	The form elements are passed in the body of the HTTP request.
Size	The parameter data is limited (the limit depends on the container)	Can send huge amount of data to the server.
Usage	Generally used to fetch some information from the host.	Generally used to process the sent data.

HTTP Request:

- When you use GET method, HTTP request has only request line and HTTP header.
- Where as POST, it has request line and HTTP body.

Parameter Passing:

When you type in your yahoomail and you pass in your username and password in that yahoomail screen. So how are these username and password passed to the Server? There are passed in Form Elements or Form Parameters of the request.

- When you are for GET, these form elements are appended to the end of the URL. They are visible in the URL.
- But, in case of POST, these form elements are passed in the body of the HTTP request. HTTP body is not seen by anyone. This is a secret HTTP body

anybody cannot be seen. The form parameters here are sent in HTTP body not in the URL which can be seen by everyone.

Size:

- Although, you can send information using GET, appending it to the end of URL; amount of data you can send is limited. Very less amount of data when implementing a GET.
- But where as, in POST you can send huge amount of data, because you send the data in the HTTP body not in the request line.

Usage:

- Basic thumb rule is, whenever you want to fetch from Server, you got for a GET.
- Whenever you want to post something to the Server use POST method.

Lets a take simple example,

You want to search for Servlets in Google Search. So you go to google.com and you type in Servlets in the search text there and you type on search. Server actually gets you back with all the search results of the term Servlets.
So here you are trying to fetch from the Servlet. In this case you should use HTTP GET.

Another example,

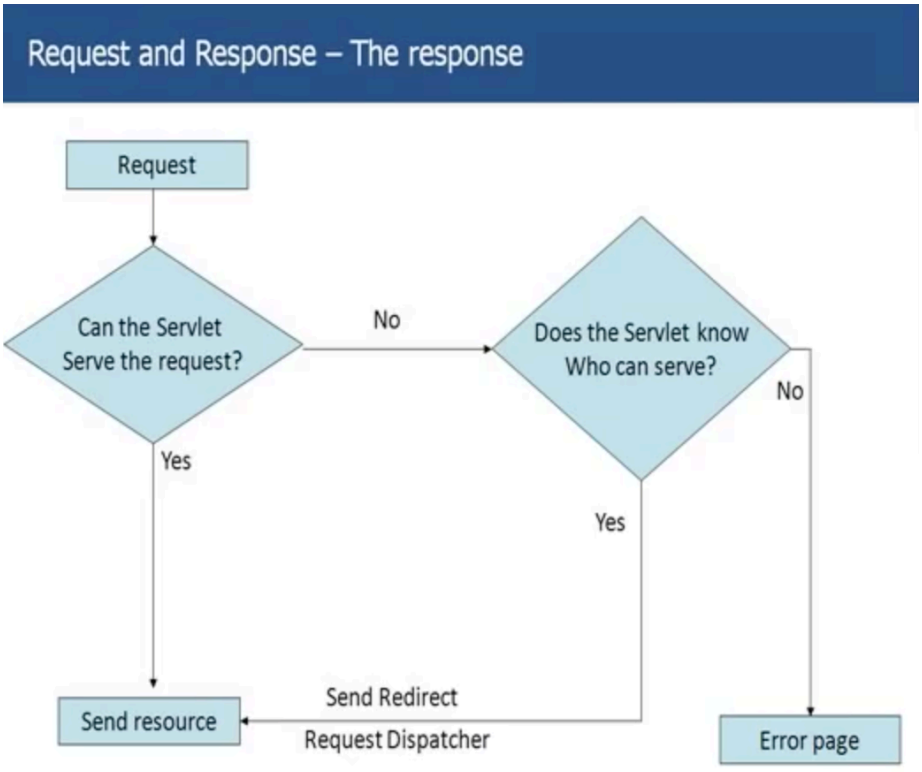
You are filling an online application form, where you enter lots of fields and what not. And submit back form to the Server. So, in this you are filling the form, submitting the data to the server. When you want to submit to the Server, we use POST method.

But, there is one more condition for this. You know that using GET, you can pass the Parameters to Server as well, but if the type of parameters is sensitive, then even though the amount of parameters sending is less you should go with POST method.

Let me say an example:

You are trying to get your emails, so you are passing username and password in yahoo logon screen and click on log on button. That will get all your emails. So, here you are trying to get emails from the Server, you should be using the POST, because the type of information you are sending is sensitive information.
When you go for the GET, username and password are appended to URL, where every one can see this username and password. And you don't want anyone to see your username and password you are using. So this information is sensitive. So all these sensitive transactions you should definitely use POST method.

The Response:



Now let's discuss about the response. So, we know that when you send request to a servlet, the container invokes the servlet and servlet sends back the response to the client.

Assume Servlet doesn't have the ability to serve the request of the client. What happens?

The Servlet cannot serve the request, now what happens?

Let's take an example, I have created a website called, abc.com and I have got lot of users on that website now. After a year, I decided to change the name of the website to xyz.com. I developed a lot of clients over this year, and I don't want to loose those clients. What I want to do is, now abc servlet cannot serve the request for the clients, but whenever a request comes to abc.com Servlet, I want the request to redirect to xyz servlet. So, I don't want the client to get an error page.

My abc servlet cannot serve the request but, does the servlet who can serve the request. Yes, it knows that xyz servlet can serve the request. Now, sends the response back to the client. It talks to xyz servlet and sends required response back to the servlet.

There are 2 ways, we can do it:

- Send Redirect: The servlet actually sends back the response to the client and client here is nothing but the web browser saying that, I cannot serve the request, but xyz.com can serve the request, please send a request to xyz.com. Web browser now, sends the request to xyz.com and gets the response successfully. You being an end client you don't response has come back from abc, and the web browser has again sent the request to xyz.
- Request Dispatcher: In Request Dispatcher, basically abc Servlet itself forward the request to xyz servlet. And response send back to the client.

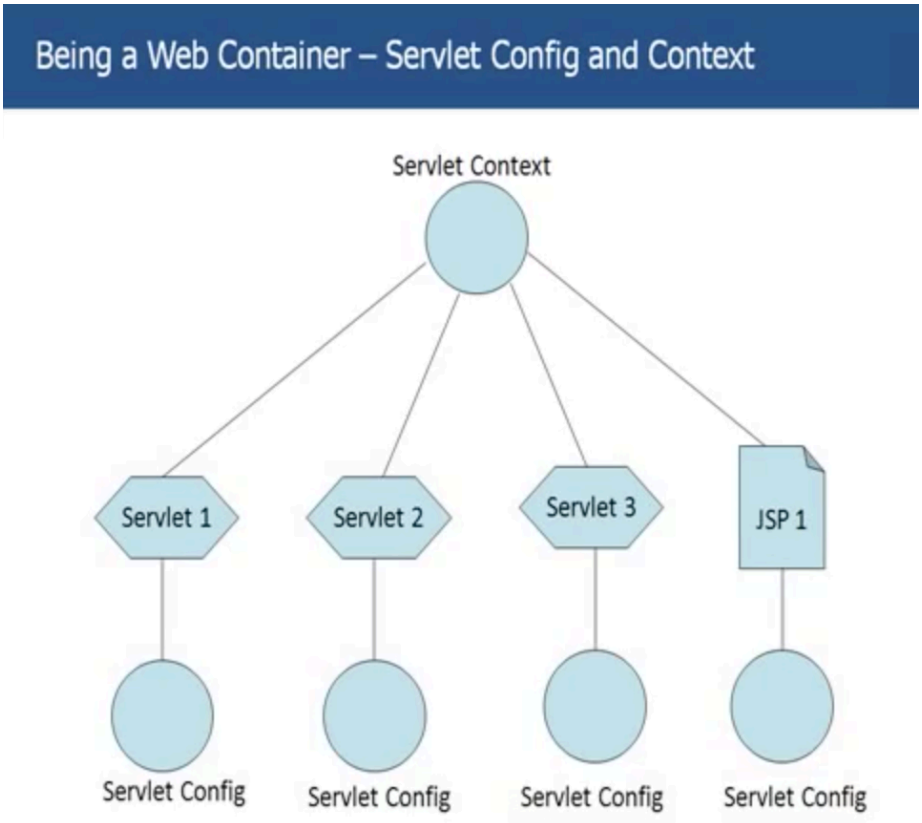
When to use Send Redirect and when to use Request Dispatcher?

You can use RD only when the other servlet which can serve the request of the client lies in the same web application. If the other servlet is in same application only in that scenario we have to use Request Dispatcher.

But other servlet is outside this web application, means in other web application then we have to use the send redirect. Because, it has to make another request.

The Servlet abc doesn't know who can serve the request sent by the client, it simply sent an error page to client.

Being Web Container - Servlet Config and Context:



Earlier we have discussed that, in the init() of the Servlet, the Servlet has got access to 2 special objects called Servlet Context and Config.

Let's have a look how that looks:

A Servlet Context is a Java Object as well as Servlet Config.
Servlet Context is only one for whole web application. And Servlet Config is one for Servlet or for JSPs. All the Servlets in the application access the same Application Context. But each Servlet has got its own Servlet Config object.

Each and every Servlet has access to Servlet Context, but Servlet 2 doesn't have access to Servlet Config of Servlet 1.

What can you do with Context and Config objects?

One thing you can definitely do is fetch your initialization parameters.

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

Comment [Office1]: http://www.javatpoint.com/servletconfig

What are init parameters?

Being a Web Container – init parameters

- What are init parameters?
- Difference between Servlet Context and Config Init parameters

	Context Init Parameters	Servlet Init Parameters
Scope	Scope is Web Container	Specific to Servlet or JSP
Servlet code	getServletContext()	getServletConfig()
Deployment Descriptor	Within the <web-app> element but not within a specific <servlet> element	Within the <servlet> element for each specific servlet

We see previously, in the web.xml slide; you have a web-app element, in that you have number of Servlet and Servlet-mapping elements and I didn't tell you that apart from these 2 elements, you will have number of other elements as well in the web.xml within the web-app element. One of those are init parameter elements.

Init parameter elements basically whatever initialization parameters you want to put in your servlets or your app basically put that initialization parameter in web.xml in the init-parameter tag.

For ex: You want to store the part to the data source the data source URL. You can put that data source URL in init-parameter element.

We have 2 types of init parameters:

1. Context init Parameters
2. Servlet init Parameters

Scope:

- Scope for Context init parameter is web container.
- Scope for Servlet init parameter is Servlet or JSP.

Servlet Code:

How do you get the init parameter value in your init()?

- getServletContext() – Here you get Context Object. On this object you say getInitParameterValue();
- getServletConfig() – ConfigObject on which you call the init parameter.

Deployment Descriptor:

- Within the <web-app> element but not within a specific <servlet> element.
- Within the <Servlet> element for each specific servlet.

Attributes:

Where we can store and retrieve data using attributes as well.

What's the difference between Attributes and parameters?

We have just seen that using parameters you can store some data and retrieve some data in Servlet code.

An attribute is an object that is used to share information in a web app. Attribute allows Servlets to share information among themselves. Attributes can be SET and GET from one of the following scopes :

Comment [Office2]: http://www.studytonight.com/servlet/attribute.php

Being a Web Container - Attributes

- What exactly, is an attribute?
- Difference between Attributes and parameters

	Attributes	Parameters
Types	Context Request Session	Context Servlet Init
Method to set	setAttribute(String, Object)	We cannot set Init parameters.
Return type	Object	String
Method to get	getAttribute(String)	getInitParameter (String)

Types:

- You can have Context attributes, you can have Session attributes and You can have Request attributes. This says nothing but the life of the attributes. Context attribute is accessed through out the application, where as the request attribute is available only for that particular request. Session attribute is valid until the session is live.
- Context Parameter and Servlet Init Parameter.

Method to Set:

- setAttribute(String, Object) – This is similar to HashMap. Key-value pair. Basically on what object you call the setAttribute(), whether call on Session, Request or Context object.
- We cannot set init parameter dynamically. We can only set once, that in web.xml. This is one major difference.

Return type:

- Object – You can store any Objects
- String – You can only store String in Parameters

Method to get:

- getAttribute(String)
- getInitParameter(String)

If you want to put in something which is used once through out the application and if that is String, you can use init Parameter.

If something that change frequently, and which is an object you go for attributes.