**Servlet , ServletConfig**

**|**

**Generic Servlet**

**|**

**HttpServlet**

**|**

**MyServlet**

**javax.servlet.GenericServlet**

* It’s an abstract class, part of Servlet API a sub-class of GenericServlet is called as Servlet & it can handle “ANY Protocols” including HTTP & HTTPS protocols
* In other words , It becomes “Protocol –Independent Servlet”
* GenericServlet has “one abstract method” ,by name **service**(), hence it’s an “abstract class”

Syntax:

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

* Hence whenever we create Servlet by extending GenericServlet we MUST provide an implementation for service(SR,SR) method.

**Javax.servlet.http.HttpServlet**

* A sub-class of HttpServlet is called as Servlet & it can handle ONLY HTTP & HTTPS protocols.
* In other words, It becomes “Protocol-Dependent Servlet”
* It’s an abstract class but none of the methods in this class are declared as abstract.
* The implementation is
* It checks whether request came via HTTP/ HTTPS protocol
* If No, then it throws an ServletException with “non-HTTP request or response”
* If YES, then it invokes “Overloaded version of service method i.e service(HSR, HSR)
* Implementation present in service(HSR, HSR) is
* It gets the HTTP Method Present in the Request Object.
* Invokes one of the corresponding doXXX(HSR,HSR) method by passing request response objects
* If request has HTTP Method as CONNECT, then this methods return “Method is not suppported by the Servlet API” error response.
* All the 7 doXXX(HSR,HSR) methods in HttpServlet has the logic of Generating “405 Error Response”
* A subclass of HttpServlet can override any of the below service() method

1. Pulbic void service(ServletRequest req, ServletResponse res) throws ServletException, IOException
2. Public void service(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
3. Protected void doXXX(HttpServletRequest req, HttpServletResponse res) throws ServletException , IOException

* We Should not override the first two versions of service methods & Our job is to override one/more doXXX(HSR,HSR) method(s)
* If we won’t override doXXX() methods, then default implementation from HttpServlet is invoked which in turn return “405 Error response”

**Why HttpServlet is an Abstract Class?**

* HttpServlet does not have any abstract methods. But being “abstract”, we are **forced to subclass**
* So Subclass can either

-inherit doXXX(HSR, HSR) methods OR

Override doXXX(HSR, HSR) methods.

* It’s Servlet Developer choice & this choice is available “ONLY being an abstract Class with Zero methods as abstract”
* Thus, to force us to implement our own servlet class, the HttpServlet class is marked as abstract.

Difference between GenericServlet & HttpServlet

|  |  |
| --- | --- |
| Generic Servlet | HttpServlet |
| Protocol Independent | Protocol Dependent. Supports only HTTP & HTTPS protocols. |
| Abstract Class; because service() method is declared as abstract | Abstract class; but none of the methods are declared as abstract |
| If we extend the GenericServlet then we must provide the implementation for service() method | There is NO restriction on overriding any version of the service method but generally we override one or more doXXX() methods. |
| GenericServlet does not extend any other Servlet API related class | HttpServlet extends GenericServlet which is part of Servlet API |
| GenericServlet implements Servlet API related interfaces such as “Servlet” & “ServletConfig” | HttpServlet does not implements any Servlet API related interfaces. |

**Assingment 4:**

**Create a HTML Form as shown below:**

**Login.html**

Create a servlet by name “LoginServlet” which takes the request from this form,

1. Gets the Form Data
2. Authenticate the Credentials by interacting with BECME89\_DB database.
3. If in-valid Credential, then display “in-Valid User Name/Password Error Message in Browser
4. If Valid , then display corresponding Reg. No. Data (combination of Data Present in studentsinfo &guardian\_info Tables), in the Browse as shown below.

**Servlet LifeCycle:**

Lifecycle of a Servlet is controlled by Servlet Container & it has following phases

1. Instantiation Phase
2. InitializationPhase
3. Service Phase
4. Destruction Phase
5. **Instantiation Phase:**

* Whenever request comes to a container, by looking at the Url and & 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”

1. **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, etxl
* This method is called ONLY ONCE in servlet Lifecycle
* We may/maynot override this method. If we don’t override then default implemetation present in GenericServlet is invoked
* The first line of the Version 1 init method should be “super.init(config)”
* **During initialization Servlet has access to following two key objects**

1. **Javax.servlet.ServletContext**
2. **Javax.servlet.ServletConfig**

**Can we use Constructor for Initialization?**

* We can also make use of Constructor for initialization but this approach is not so common
* Also , init method has access to ServletContext & ServletConfig objects where as Constructor don’t
* So if initialization code is dependent on these two objects then we have to make us of init method. Also in case of constructor , we must make use of “Public Default Constructor”
* Hence we generally make use of init method for initialization.
* Once instantiation & initialization is successful, container caches the Servlet Instance

1. **Service Phase:**

public void service(ServletRequest req, ServletResponseres) 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 Lifecycle
* If a Servlet is a sub-class of GenericServlet then we MUST override tis 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() methods.

4. **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 invokes 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 LifeCycle
* We may/may not override this method. If we don’t override then default implementation present in GenericServlet is invoked,

**NOTE:**

No matter how we create a Servlet, Container ALWAYS invokes below Lifecycle Methods on that Servlet

1. public default Constructor
2. void init(ServletConfig)
3. void service(ServletRequest, ServletResponse)
4. void destroy()

**Note:**

* Any Class which extends any one of the below class is called as a “Servlet”

javax.servlet.HttpServlet

javax.servlet.GenericSevlet

* **If** a class extends either HttpServlet or GenericServlet the “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 “instantiating Phase”.**
* Servlets MUST have public default Constructor Or combination of any other constructor along with public default constructor.
* There is only one instance exist for any servlet. i.e. Servlets are “Singleton in nature”.
* Servlets are protocol independent in nature but are most often used with HTTP & HTTPS protocols.
* 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 Application are “Multi Threaded environments**”
* **Marker interface has empty body.**
* **1. java.io.Seralizable**

**2 . java.lang.Cloneable**

1. **Java.util.RandomAccess**
2. **Java.rmi.Remote**
3. **Java.util.EventListener**

**Servlet API**

1. Javax.servlet.SingleThreadModel.

**Single Threaded Servlets:**

* We know that , by default servlets are Multi Threaded in Nature. Hence following are the 2 ways to create Single Threaaded servlet

1. By Implementing “javax.servlet.SingleThreadModel” Marker Interface Or
2. By Synchronizing Service Method.

* SingleThreadModel is a Marker Interface which ensure that servlets handle only one request at a time i.e. Container start handling the requests “Synchronously”
* This interface is “Deprecated” in Servlet API 2.4

**Example:**

public class MyServlet extends HttpServlet/GenericServlet implements SingleThreadModel

{

//Servlet code Goes Here

}

**Assignment 6:**

>Create a HTML Page with Hyper-Link as Shown below

AllStudentsView.html

Create a Servlet which gets the request from this Hyper-Link & display All the students information(Combination\_info table) & display the data in browser as shown below.

**Diffrences between ServletContext & ServletConfig:**

|  |  |
| --- | --- |
| **Servlet Context** | **Servlet Config** |
| SevletContext is an Interface and “an Object of ServletContext” is used by container to pass information to ALL the servlets which are part of an application | ServletConfig is an Interface and “an Object of ServletConfig” used by a container to pass information to a particular Servlet. |
| ServletContext object is created at the time of Server Startup & Garbage collected during the server shutdown | ServletConfig Object is created during the “initialization Phase” of Servlet Lifecycle & Garbage collected during “Desstruction Phase”. |
| So there will be “ONLY ONE Instance of ServletContext object exists per web application” | There will be “ONLY ONE Instance of ServletConfig object exists per Servlet” |
| Hence “Singleton” in Nature | “Non-Sighleton” in Nature |
| ServletContext object is obtained by calling “getServletContext()” method which we inherit from GenericServlet | ServletConfig object is obtained by calling “getServletConfig()” method which we inherit from “GenericServlet” |
| ServletContext context =  getServletContext(); | ServletConfig config = getServletConfig(); |
| In web.xml, context parameters are declared under <context-param> tag (one/more) | In web.xml, servlet config parameters are declared under <init-param> tag (one /more) which is a subtag of <servlet> tag. |
| ServletContext object “does not “ holds the object reference of ServletConfig  ServletContext context = config.getServletContext(); | ServletConfig object holds the object reference of ServletContext  ServletConfig config = getServletConfig(); |

**Note:**

1. Both ServletContext & ServletConfig objects has a method by name getInitParameter() which helps us to get parameter value information from web.xml

Syntax:

String getInitParameter(String paramName)

1. **We can ONLY get ServletContext & ServletConfig parameters at Runtime but “we can not set them”**