Java Servlets 3.0

Lesson 03: Introduction to Servlets API and Ease of Development through Annotations

Lesson Objectives

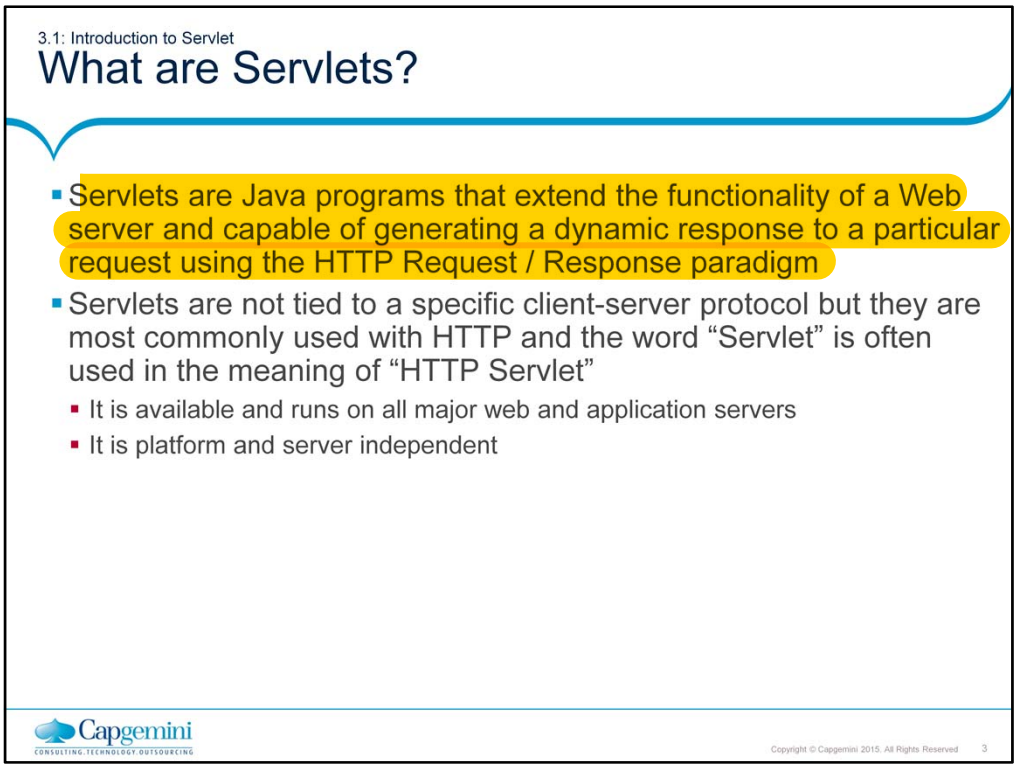
- In this lesson, we will learn:
  - Introduction to Servlet
  - Role of Servlets in Web application design
  - Advantages of Servlets
  - HTTP Basics
  - Basic Servlet Architecture : Servlet Container
  - Servlet Lifecycle
  - Ease of Developing Servlets through Annotations
  - Servlet Configuration and Accessing Initial/Context Parameters via Annotations
  - Retrieving Information

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

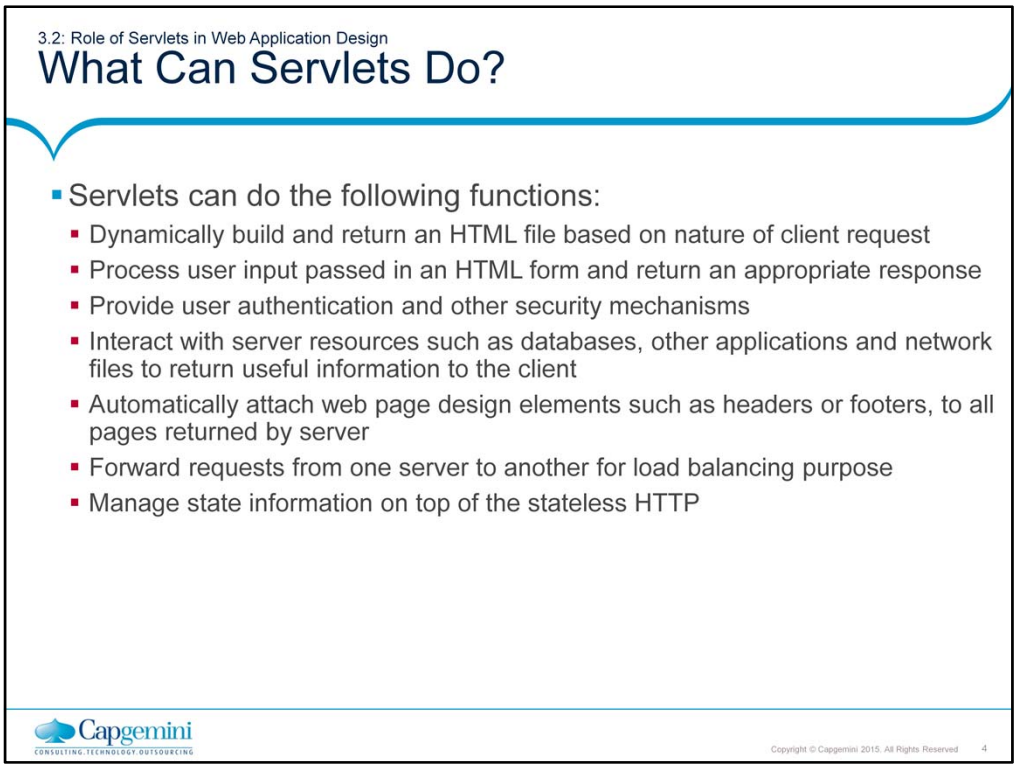
Copyright © Capgemini 2015. All Rights Reserved 2

Lesson Objectives:
This lesson introduces Servlets 3.0 and ease of development through Annotations , The lesson contents are:

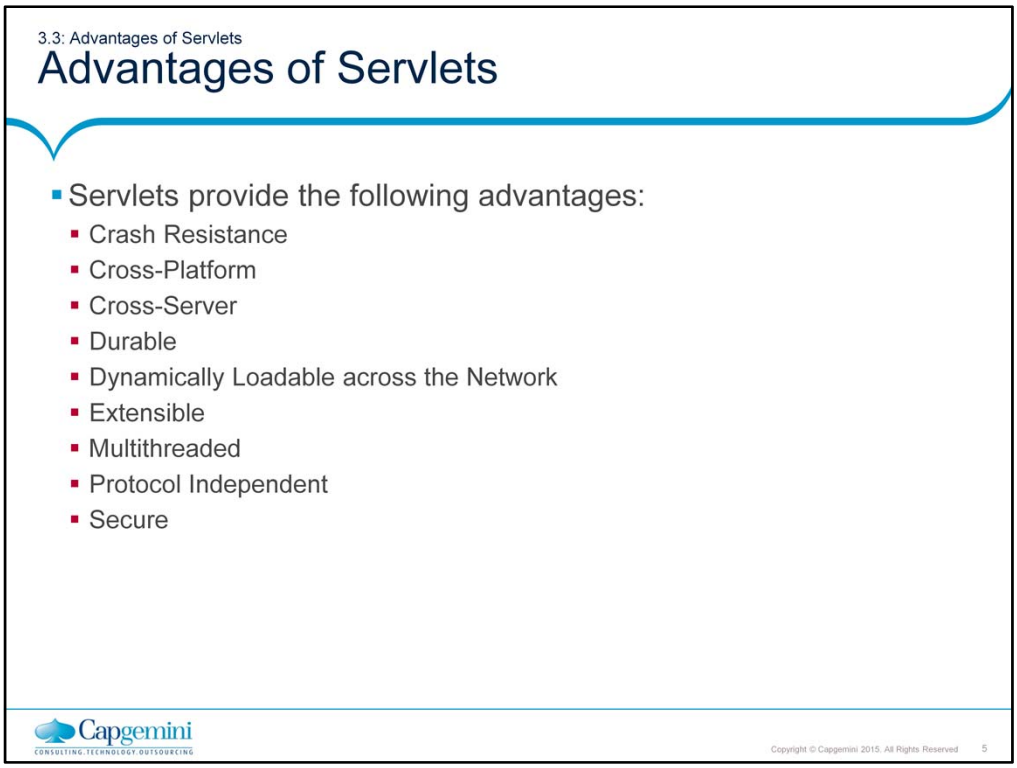
3.1: Introduction to Servlet
3.2: Role of Servlets in Web Application Design
3.3: Advantages of Servlets
3.4: HTTP Basics
3.5: Basic Servlet Architecture: Servlet Container
3.6: Servlet Lifecycle
3.7: Elements of Web Application
3.8: Ease of Developing Servlets through Annotations
3.9: Servlet Configuration and Accessing Initial / Context Parameters via Annotations
3.10: Retrieving Information



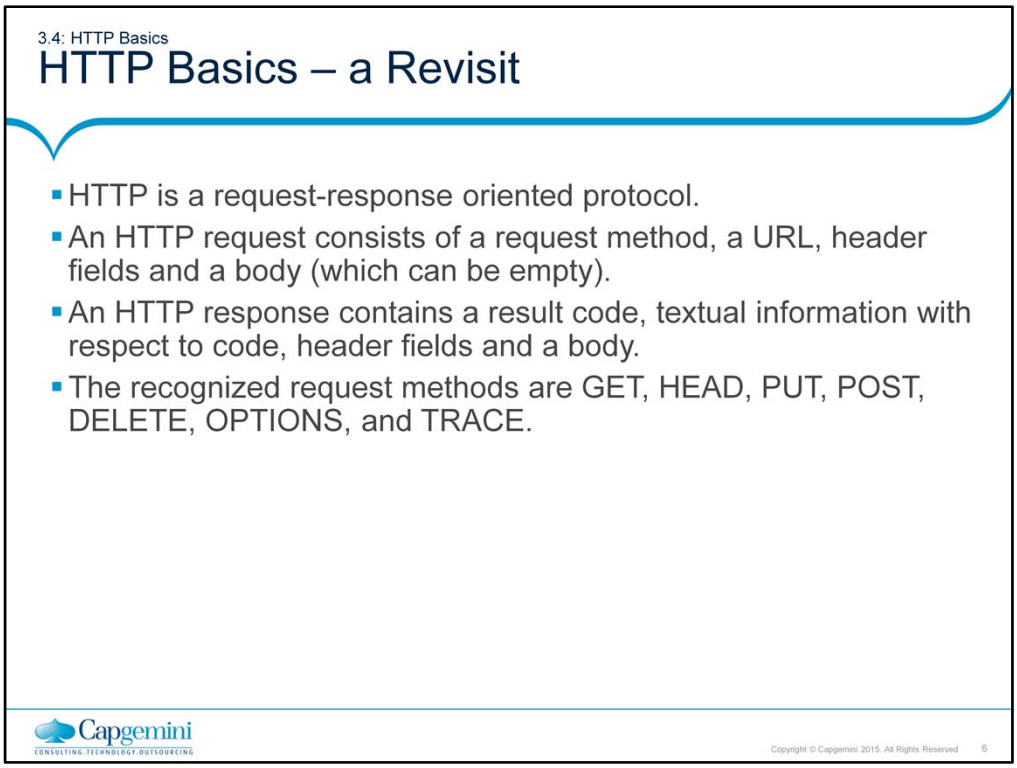
Introduction to Servlet:
Java servlets are a key component of server-side Java Development. Servlets are modules of Java code that run in a server application to answer client requests. These are small, pluggable extensions to server that enhance the server's functionality.
Servlets allow developers to extend and customize any Java-enabled server.
When servlets are used to generate dynamic content for a web page or otherwise extend the functionality of a web server, is like creating a web application!
Since servlets are written in the highly portable Java language and follow a standard framework, they provide a means to create sophisticated server extensions in a server and operating system independent way.
Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes. We shall see more on this in the coming sections.
Servlets first came on the scene around 1997. As of March 26 2010, the current version of the servlet specification is 3.0. The minimum platform requirement for Servlet 3.0 is JDK 1.6.



Role of Servlets in Web Application Design:
Servlets can read explicit data sent in by the client in browser. This data can come from an HTML page into which user has entered data. Servlets can also read implicit request data which is sent by browser as part of request header.
Servlets can dynamically generate response and send content back to client. This process may involve talking to databases, executing another server-side component, and so on.
The response sent can be in from of pure HTML, plain text, XML, GIF images, or even as compressed data.



Advantages of Servlets over CGI:
Servlets provide the following advantages:
Compiled: Servlets are compiled into Java byte-codes. This improves performance through compile-time code optimization. Server-side JIT compilers dramatically improve the performance of JVM.
Compilation also offers the advantages of strong error and type checking. Since many errors are flushed out during the compilation, servlets are more stable and easier to develop and debug.
Crash Resistance: The JVM does not allow servlets direct access to memory locations, thereby eliminating crashes that results from invalid memory accesses. In addition, before execution, the JVM verifies that compiled Java class files are valid and do not perform illegal operations. Finally, rather than crashing, the JVM will propagate an exception up the calling chain until it is caught. Thus, a poorly written or malicious servlet cannot crash the server.
Cross-Platform: Since servlets are written in Java, they enjoy the same cross platform support as any program. This “write once, run anywhere” capability allows servlets to be easily distributed throughout the enterprise without rewriting for each platform.
Cross-Server: Servlets can be run on virtually every popular web server that is in use today. More than a dozen software vendors currently provide native support for servlets within their products. For those servers that do not currently offer native servlet support, there are many third party add-ons that allow these servers to load and run servlets.

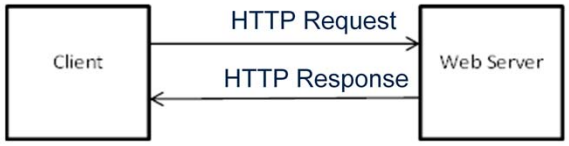


Basic Servlet Architecture:
HTTP Basics – a Revisit:
HTTP is the protocol that is used by a WWW client (for example: a browser) to send a request to a Web Server.
HTTP is a request-response oriented stateless protocol. An HTTP request consists of a request method, a URL, header fields, and a body (which can be empty). An HTTP response contains a result code, textual information w.r.t code, header fields and a body.
When client request a URL in a Web Browser, the default method is GET for the request. With a GET request, the input parameters are appended to the URL. A GET request does not have a body.
With a POST request the input parameters are transmitted in the body.
The response should contain a body with the response data and header fields, which describe the body (especially Content-Type and Content-Encoding).

3.5: Basic Servlet Architecture : Servlet Container

Servlet Container

- What is a Web Server?
  - To know what is a Servlet container we need to know what is a Web Server first



- A web server uses HTTP protocol to transfer data. In a simple situation, a user types in a URL (e.g. www.servletdemos.com/static.html) in browser (a client), and get a web page to read. So what the server does is sending a web page to the client. The transformation is in HTTP protocol which specifies the format of request and response message.

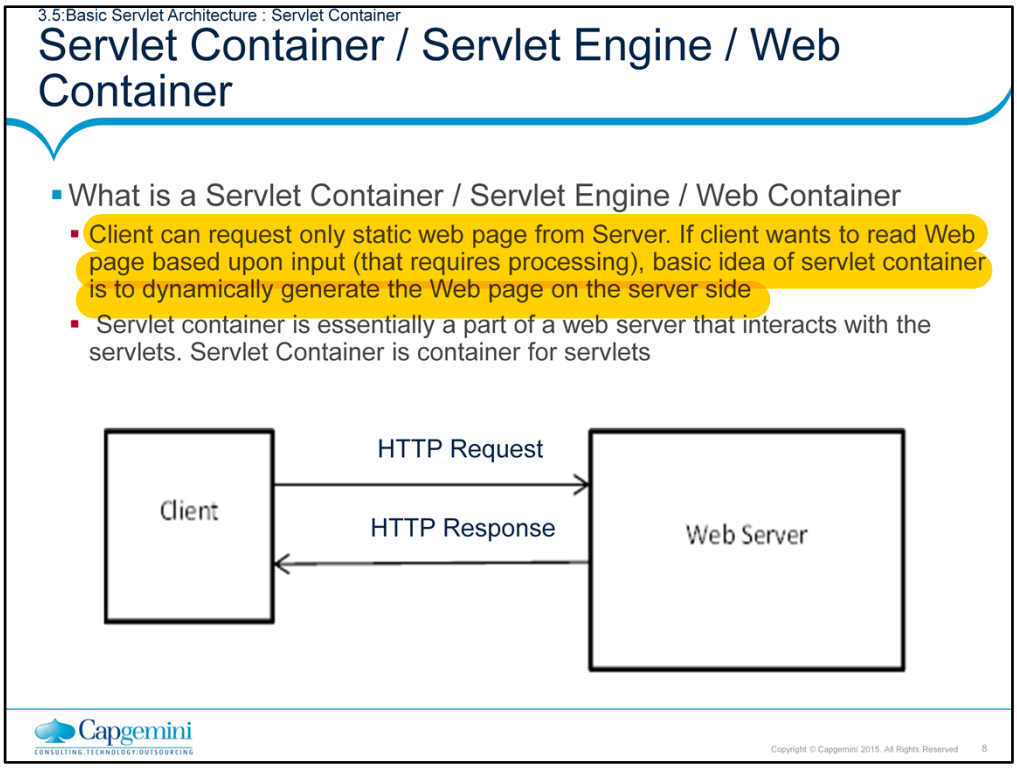
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 7

Since we using WildFly 8.x which is an Application server which will host the Servlet Container to handle the web components.

We could also use the Web server for the same purpose.

Since HTTP is a web oriented protocol, thus we have taken the term to be as “Web Server”



When a request is sent, it is handled by the web server. If response is available within the web server it will be returned by the server to the client.

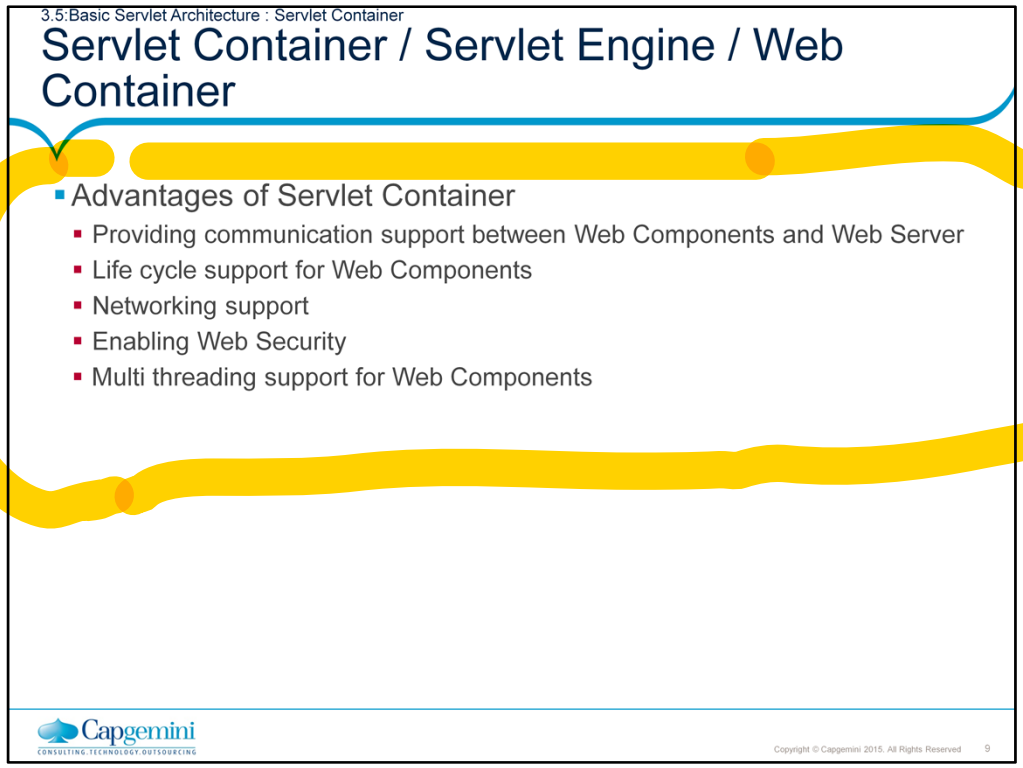
If response is not available and requires some processing then request will be processed by the Web Container.

The web container will then process the request and handover the response back to web server. The web server will then handover response back to client.

The web container delegates the request processing to web components like Servlets and JSP.

For example: If we request for a page as www.gmail.com , then same page is returned as response for all users. **This is a static content**

Whereas after entering your login credentials respective inbox details per user is retrieved as a response. **This is dynamic content**



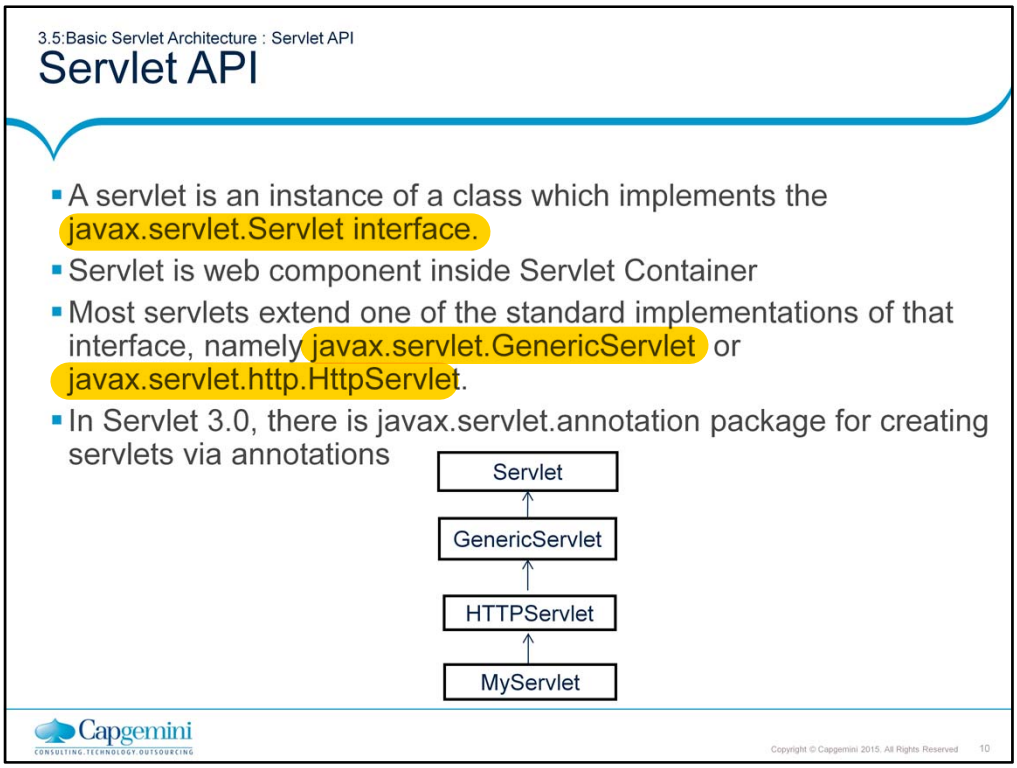
Providing communication support between Web Components and Web Server: Servlet as a web component does not communicate with Web Server directly. All the communication happens via Web container

Life cycle support for Web Components: Life cycle of Web components (Servlets and JSP) is managed by the container

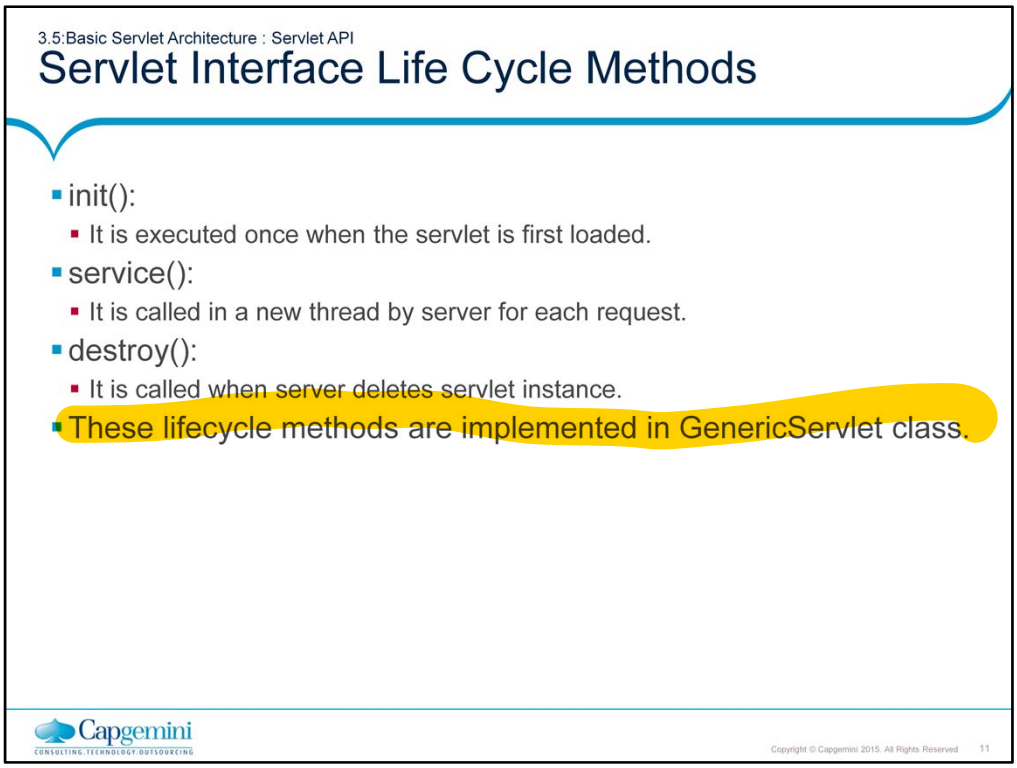
Networking support: Web components do not need to open network and socket connections all is taken care by Web server

Enabling Web Security: Access to Web components can be secured by granting privileges.

Multi Threading support: Web components are multi-thread, for every request there is one thread generated. Thus all threads are managed by container. We do not need to do explicit multi-threading.



Basic Servlet Architecture:
Servlet API:
Servlets use classes and interfaces from three packages: javax.servlet, javax.servlet.http.
A Servlet, in its most general form, is an instance of a class which implements the javax.servlet.Servlet interface. Most Servlets extend one of the standard implementations of that interface, namely javax.servlet.GenericServlet and javax.servlet.http.HttpServlet.
As of Servlet 3.0, make use of package javax.servlet.annotation to create Servlets.
A protocol-independent servlet should subclass GenericServlet, while an HTTP servlet should subclass HTTPServlet, which is itself a subclass of GenericServlet! (refer to the above figure).
Notice that the classes do not belong to the core Java API. They are extensions to the core API and hence javax!



Basic Servlet Architecture:
Servlet Interface Life Cycle Methods:
The Servlet interface defines methods to initialize a servlet, to service requests, and to remove a servlet from the server. These are known as life-cycle methods and are called in the following sequence:

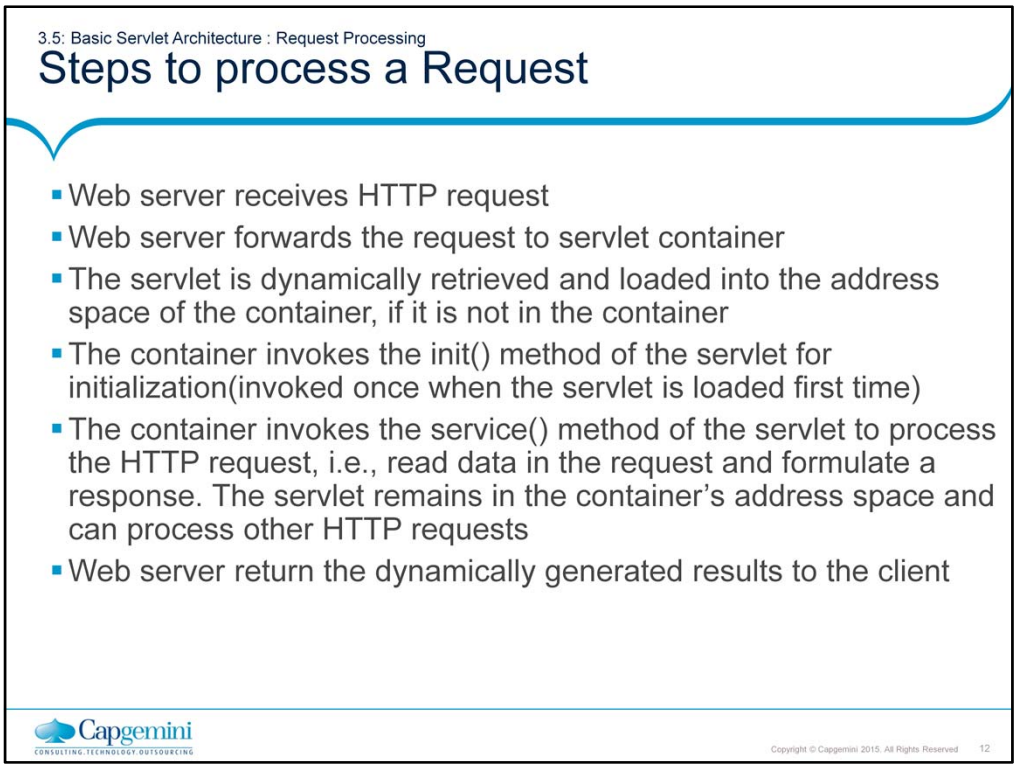
The init() method is guaranteed to be called only once during the Servlet's lifecycle. The Servlet performs one-time setup configurations in this method. It stores the ServletConfig[*] object so that it can be retrieved later by calling the Servlet's getServletConfig() method (This is handled by GenericServlet). The ServletConfig object contains Servlet parameters and a reference to the Servlet's ServletContext[*].

service() method gets called every time a new request comes in. The method is called concurrently (that is, multiple threads may call this method at the same time) so it should be implemented in a thread-safe manner.

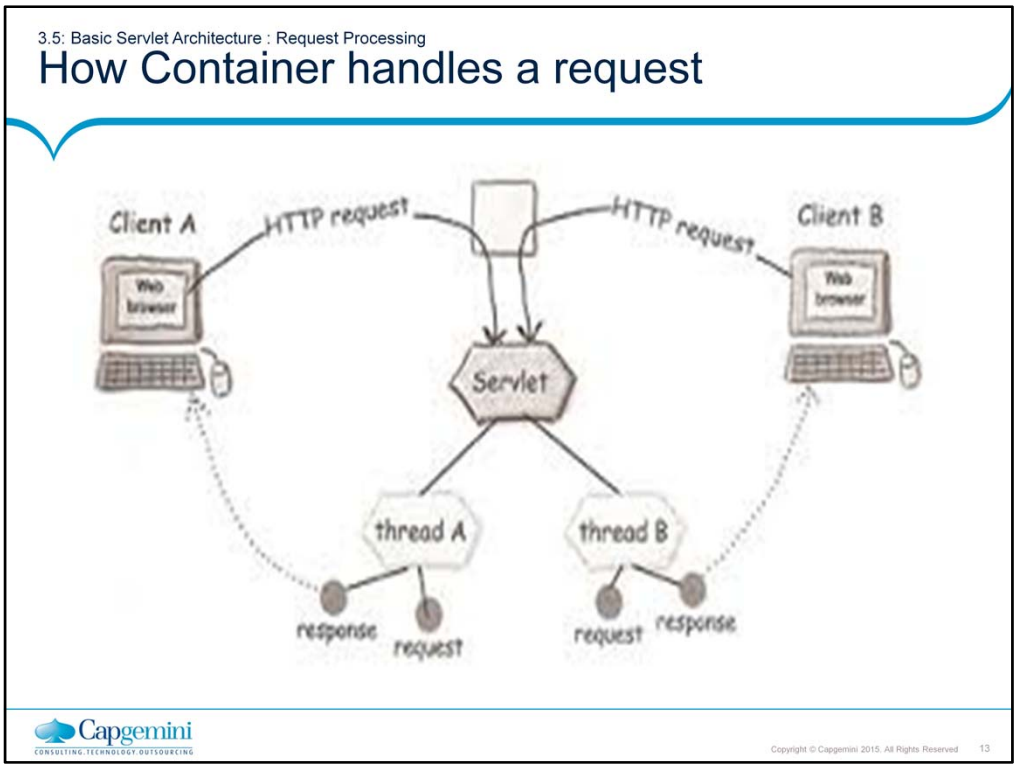
When servlet needs to be unloaded (for example: since a new version should be loaded or the server is shutting down), the destroy() method is called. This method too is guaranteed to be called only once during the Servlet's lifecycle. All resources which were allocated in init() should be released in destroy().

We shall be covering entire lifecycle of a typical servlet in the next section.

[*] : The ServletConfig and ServletContext interfaces are explained in the next slide.



From the life cycle of a servlet object, we can see that servlet classes are loaded to container by class loader dynamically. Each request is in its own thread, and a servlet object can serve multiple threads at the same time. When it is no longer being used, it should be garbage collected by JVM.
Like any Java program, the servlet runs within a JVM. To handle the complexity of HTTP requests, the servlet container comes in. The servlet container is responsible for servlets' creation, execution and destruction.

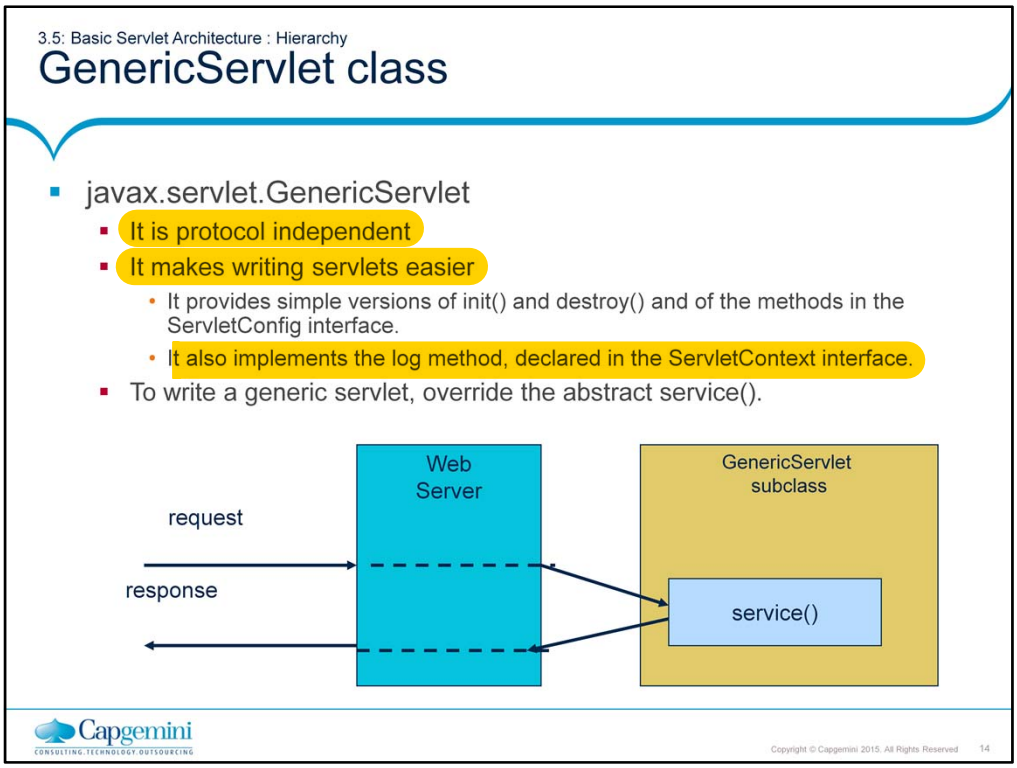


Every request coming to the container from the client for a particular servlet will be treated as a separate thread.

Each thread will have its own copy of the request / response objects.

After the processing of a particular request is completed the threads would be garbage collected. But the servlet instance would still be there in memory.

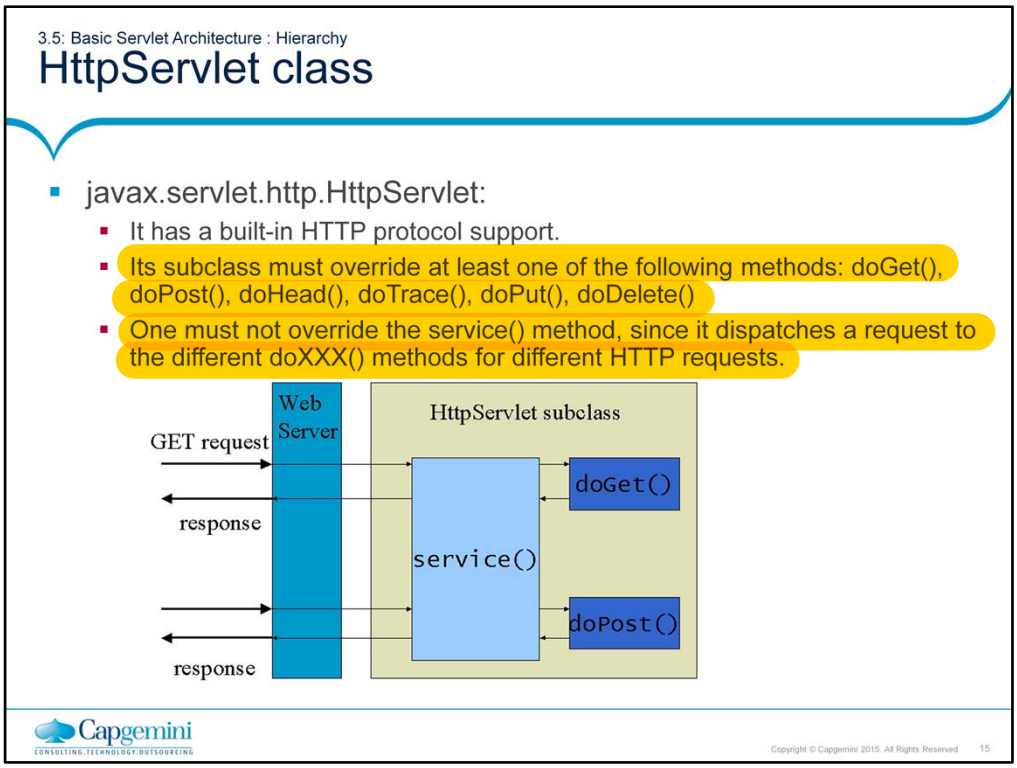
Thus for processing multiple requests multiple threads are created.



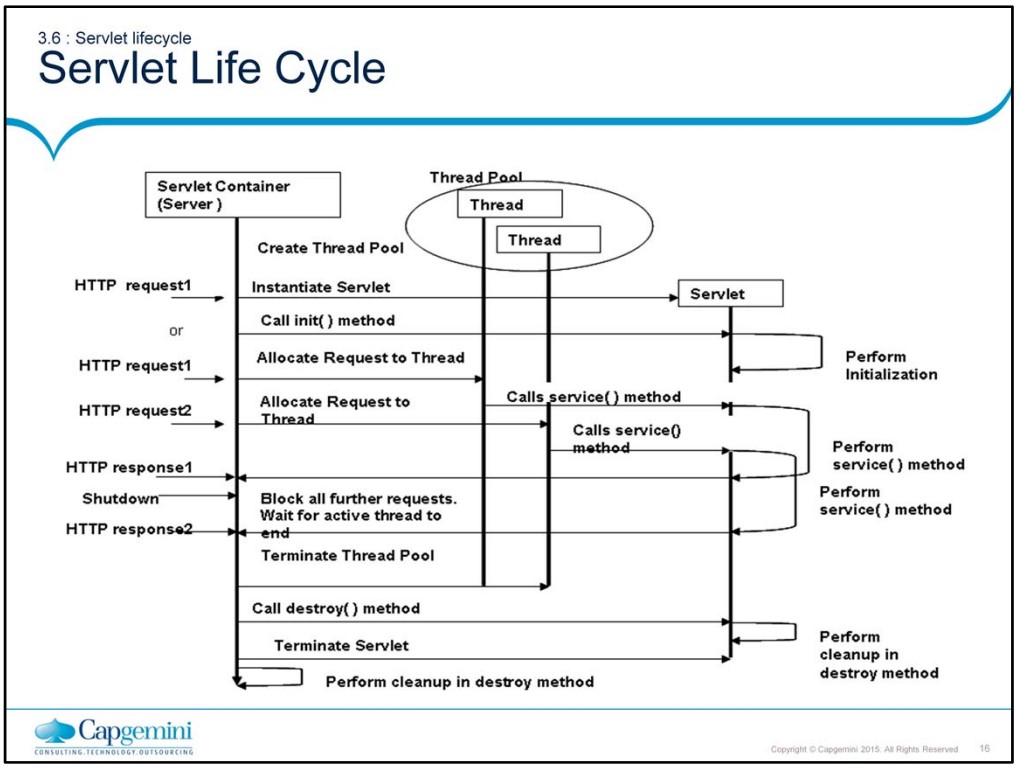
Basic Servlet Architecture:
GenericServlet:
All the lifecycle methods are implemented in GenericServlet class. Thus if GenericServlet class is extended to create a servlet, all the methods may be overridden to provide implementations. Optionally, just override the service() method to handle the request in the manner required!

ServletContext interface :
It defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.
There is one context per “web application” per JVM.
ServletContext attributes can be used to share information among a group of servlets.
The ServletContext object is contained within the ServletConfig object, which the Web server provides the servlet when the servlet is initialized.
ServletConfig interface:
This is a servlet configuration object used by a servlet container used to pass information to a servlet during initialization.
It is implemented by GenericServlet.
All of its initialization parameters can be set in deployment descriptor. Or else they can be passed via annotations. The ServletConfig parameters are specified for a particular servlet and are unknown to other servlets.

ServletContext and ServletConfig will be discussed in later sessions.

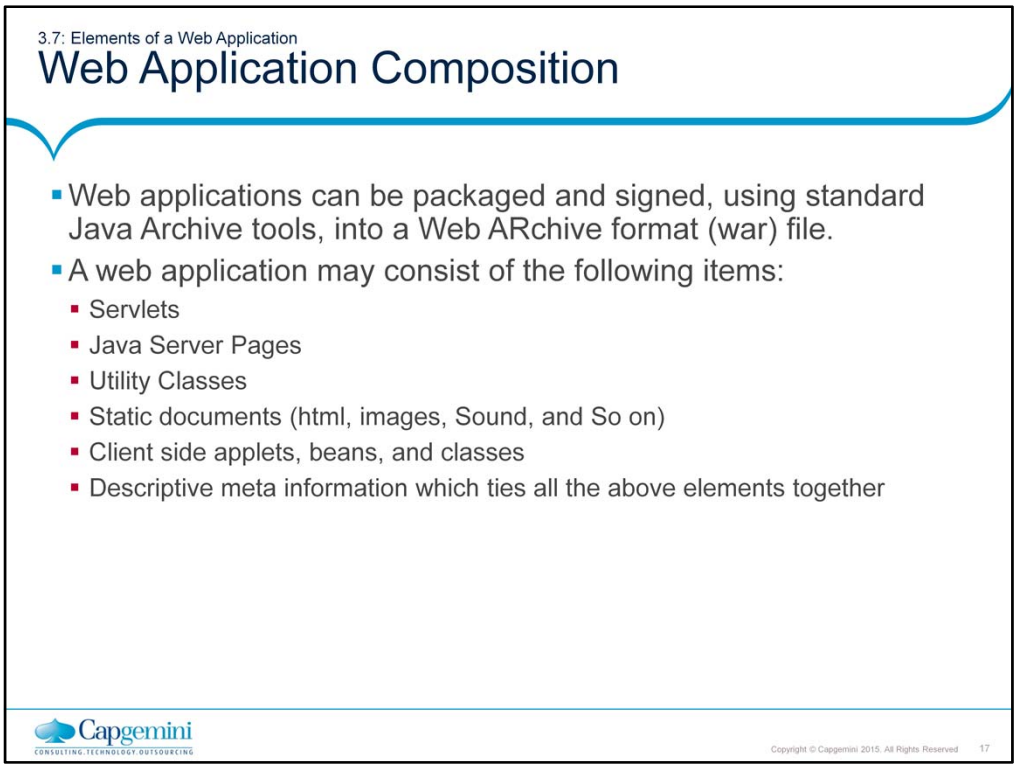


Basic Servlet Architecture:
HttpServlet:
init() and destroy() can be overridden to manage resources that are held for the life of the servlet. Servlet can implement getServletInfo() to provide information about itself.
The service() method of HttpServlet dispatches a request to different Java methods for different HTTP request methods. It recognizes the standard HTTP/1.1 methods like GET, HEAD, PUT, POST, DELETE, OPTIONS and TRACE. Other methods are answered with a Bad Request HTTP error.
Do not override the service() method because it handles setup and dispatching to all the doXXX() methods. Since service() method dispatches to doXXX() methods, it passes its request and response objects to these methods.
An HTTP servlet generally overrides doGet() to handle GET requests and doPost() to handle POST type of requests.

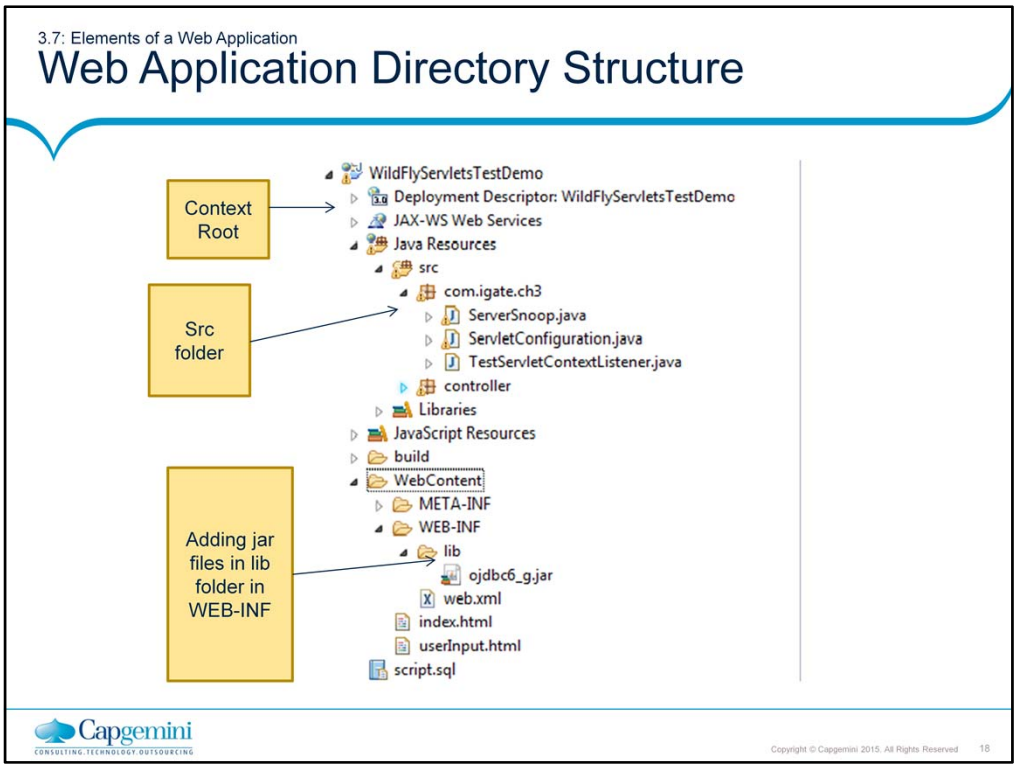


Basic Servlet Architecture:
Servlet Life Cycle:
The process by which a server invokes a servlet can be broken down into the nine steps described below and as shown in the figure in the above slide:

The server loads the servlet when the client first requests it or, if configured to do so, at server start-up. The servlet may be loaded from either a local or remote location using the standard Java class loading facility. This step is equivalent to the following code:
Class c= Class.forName("com.igate.Myservlet");
The server creates one or more instances of the servlet class. Depending on the implementation, the server may create a single instance that services all requests through multiple threads or create a pool of instances from which one is chosen to service each new request. This step is equivalent to the following Java code:
Servlet s= (Servlet) c.newInstance();
The server constructs a ServletConfig object that provides initialization information to the servlet.
The server calls the servlet's init(ServletConfig cfg) method. The init() method is guaranteed to finish execution prior to the servlet processing the first request. If the server created multiple servlets instances (step 2), then the init method is called one time for each instance.



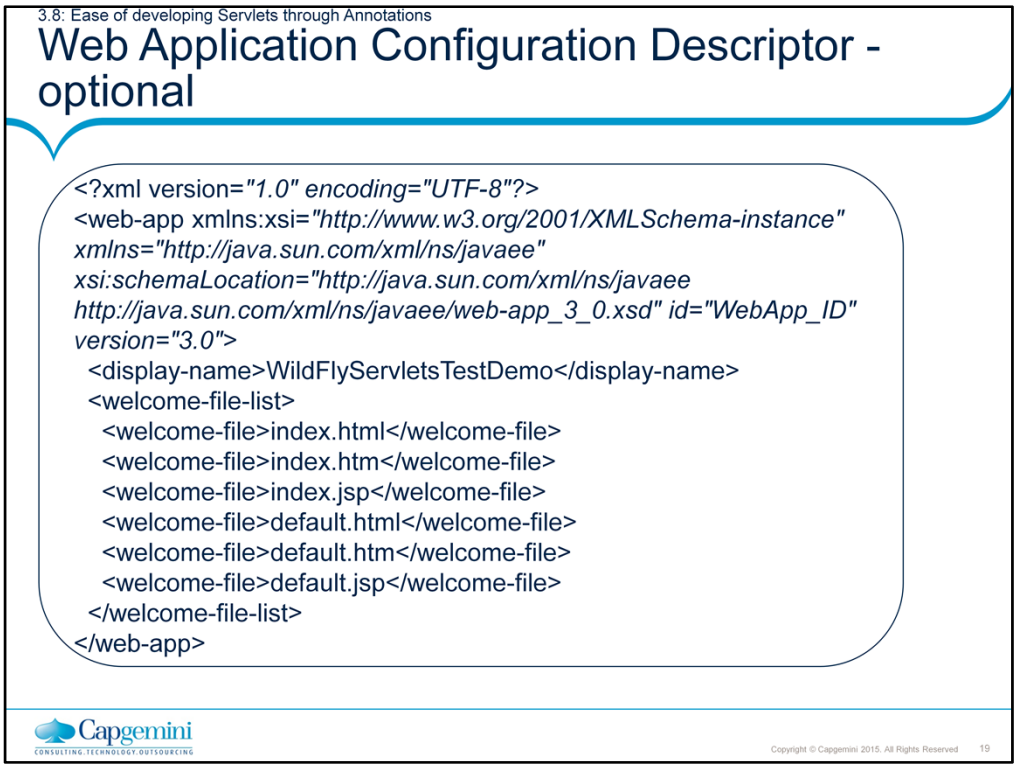
Elements of a Web Application:
Web Archive (WAR) files provide a convenient way of bundling Web applications in a single file. Having a single file instead of many small files makes it easier to transfer the web application from server to server.
A WAR file is really just a JAR file with a .war extension. We can use the jar command to create it.
For example: jar cvf myWebApp.war *
However, we can also use the ANT tool or IDE like Eclipse to automate the process!
A Web application is a collection of web elements that can be bundled and run on multiple containers (servers) from multiple vendors.
A web application is rooted at a specific path within a web server called context root (see figure on the next slide).
For example, a catalog application can be located at http://www.mycorp.com/catalog. All requests that start with this prefix will be routed to the ServletContext, which represents the catalog application.
The servlet container must enforce a one to one correspondence between a web application and a ServletContext. A ServletContext object can be viewed as a Servlet's view onto its application.



Elements of a Web Application:
Directory Structure:
A web application exists as a structured hierarchy of directories. The root of this hierarchy (called Context Root) serves as a document root for serving files that are part of this context. For example, for a web application located at /Catalog in a web server, the index.html file located at the base of the web application hierarchy can be served to satisfy a request to /Catalog/index.html.
A special directory exists within the application hierarchy named “WEB-INF”. This directory contains all things related to the application that are not in the context root of the application. Note that the WEB-INF node is not part of the public document tree of the application. No file contained in the WEB-INF directory may be served directly to a client.
The contents of the WEB-INF directory are:

web.xml: It is a deployment descriptor. As of Servlets 3.0, the creation of this file can be omitted.
classes directory: It is for servlet and utility classes. The classes in this directory are used by the application class loader to load classes from.
lib directory: It contains all .jar files. It may contains servlets, beans, and other utility classes useful to the web application. All such archive files are used by the web application class loader to load classes.
src: It is optional and will contain source code.

See the figure given in the above slide. The context-root here is called servlets-demo, which contains all web elements for a specific web application. For example: To invoke index.html, the url will be: http://<server-name>:<port-no>/servlet-demo/html/index.html.
Note: If an IDE like Eclipse Luna 4.4 is used, the folder structure might be a bit different in the IDE!

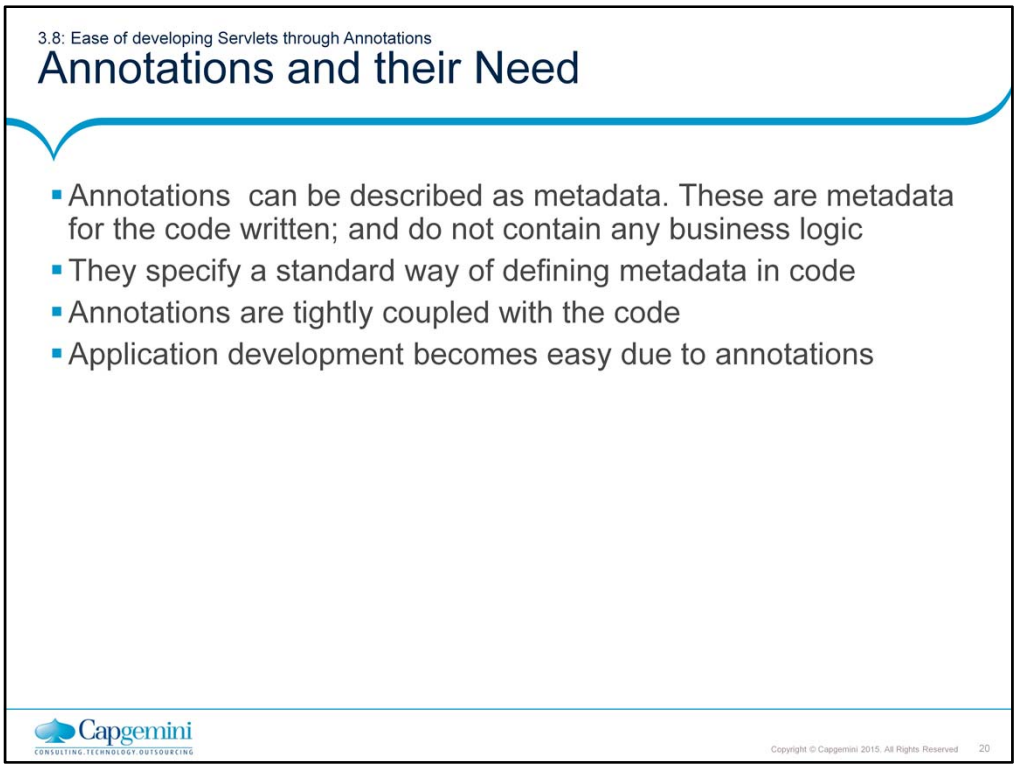


Elements of a Web Application:
Refer to Lesson 02 : To understand the steps for selecting the checkbox for web.xml

Web Application Configuration Descriptor (web.xml): is optional since Servlets 3.0
A deployment descriptor is an XML document with an .xml extension. It defines a component's deployment settings. The information provided by a deployment descriptor is declarative and therefore it can be modified without changing the source code of a bean. The Java EE server reads the deployment descriptor at run time and acts upon the component accordingly.
The web.xml file in the WEB-INF directory is a deployment descriptor. The following types of configuration and deployment information exist in the web application deployment descriptor:

- ServletContext Init Parameters
- Session Configuration
- Servlet / JSP Definitions
- Servlet / JSP Mappings
- Mime Type Mappings
- Welcome File list
- Error Pages
- Security

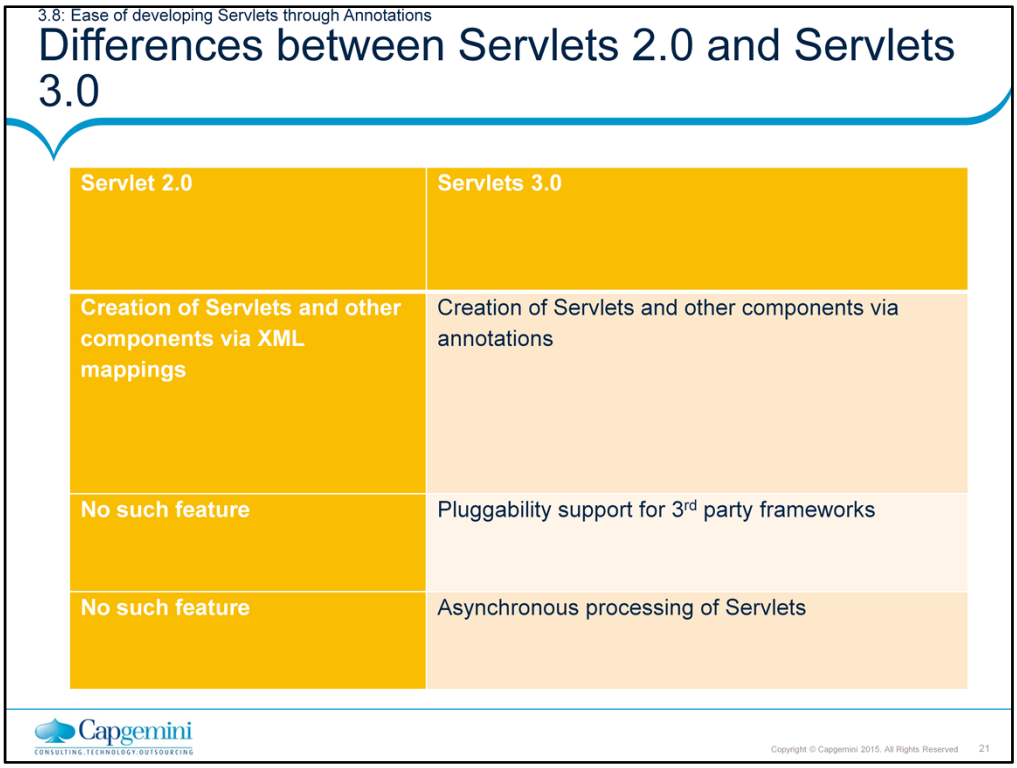
All of these types of information are conveyed in the deployment descriptor. A partial entry is shown in the code listing in the above slide.



Why Were Annotations Introduced?

Prior to annotation (and even after) XML were extensively used for metadata and somehow a particular set of Application Developers and Architects thought XML maintenance was getting troublesome. They wanted something which could be coupled closely with code instead of XML which is very loosely coupled (in some cases almost separate) from code. Interesting point is XML configurations were introduced to separate configuration from code.

Suppose, you want to set some application wide constants/parameters. In this scenario, XML would be a better choice because this is not related with any specific piece of code. If you want to expose some method as a service, annotation would be a better choice as it needs to be tightly coupled with that method and developer of the method must be aware of this.
Another important factor is that annotation defines a standard way of defining metadata in code. Prior to annotations people also used their own ways to define metadata. Some examples are – using marker interfaces, comments, transient keywords etc. Each developer decided his own way to decide metadata, but annotation standardized things. These days most frameworks use combination of both XML and Annotations to leverage positive aspects of both.



Pluggability Support for 3rd Party frameworks and Asynchronous processing of Servlets - details of which are shared in the Appendix.

Creation of Servlets and other components via XML mappings – details of which are shared in the Appendix.

In the subsequent slides we would be seeing creation of Servlets and other components via annotations.

3.8: Ease of developing Servlets through Annotations

URL Mapping of Servlets and Annotations

- @WebServlet Annotation: javax.servlet.annotation.WebServlet is a class-level annotation that affirms the annotated class as a servlet and holds metadata about the declared servlet
- The urlMappings attribute is a mandatory attribute of @WebServlet that specifies the URL pattern that invokes this servlet
- Here is the much simplified version written to the Servlet 3.0 API.
- As MyServlet is annotated as a servlet using the @WebServlet annotation, it gets initialized during the startup of the web container. Note that the deployment descriptor is optional in this case.

```
@WebServlet(name = "Basic ", urlPatterns={"/MyApp"}, loadOnStartup=1)
public class MyServlet extends HttpServlet{
public void doGet(HttpServletRequest req, HttpServletResponse res){ .... }}
```

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 22

When a request arrives, the container matches the URL in the request with the servlet's URL Patterns and if the URL pattern matches, the corresponding servlet will be invoked to serve the request. All other attributes of this annotation are optional, with reasonable defaults.

The name attribute specifies the name of servlet. It could be different from Servlet class, URL Pattern. Rather is just a dummy name given to servlets. This attribute is not mandatory.

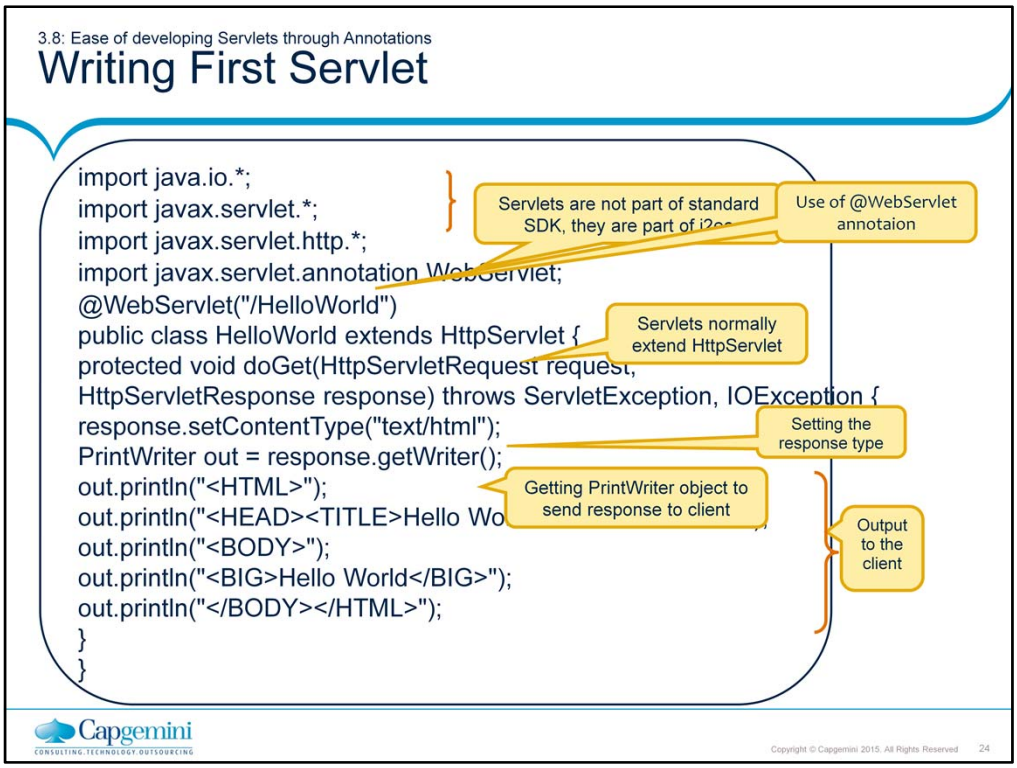
It also has loadOnStartup attribute, this is used to specify - If you want to do something on the startup of the servlet (i.e. the element loadOnStartup with value greater or equal to zero, 0), you need put the code in a init method or in the constructor of the servlet: As we know the init() method is the first life cycle method of servlet.

loadOnStartup also specifies that it will be the first servlet to execute before the client request comes for the servlet and the init() method would run.

For comparison, a code snippet for writing a Java servlet using the old Servlet 2.5 API is shown below. In Servlet 2.5, the web container will initialize the servlet only if you configure the details of the servlet in the deployment descriptor

```
public class MyServlet extends HttpServlet {
public void doGet (HttpServletRequest req, HttpServletResponse res)
{               ....

}
}
Deployment descriptor (web.xml)
<web-app>
                    <servlet>
                    <servlet-name>MyServlet</servlet-name>
                    <servlet-class>samples.MyServlet</servlet-class>
                    </servlet>
                    <servlet-mapping>
                    <servlet-name>MyServlet</servlet-name>
                    <url-pattern>/MyApp</url-pattern>
                    </servlet-mapping>
</web-app>
```



Elements of a Web Application:
The above servlet prints a simple hello world.

This code snippet is using @WebServlet annotation.
Here the URL pattern mapping is embedded within the same annotation. The class also extends HttpServlet and overrides the doGet(request, response) method.

Make use of the annotation @WebServlet above the class name which will create the servlet – HelloWorld inside the container. The servlet will now be referred by the name “HelloWorld”. The “/HelloWorld” is the URL Pattern of servlet, that would be used to invoke servlet from web browser.

URL pattern could be any logical name example: “/Hello”. It need not be necessarily same name of Servlet. A servlet could have multiple URL patterns.

The doGet() method is overridden to service all incoming requests. Line 8 uses HttpServletResponse object to set the content type of the response. Line 9 uses the response object to retrieve a PrintWriter object to send responses. The remaining lines dynamically generate the HTTP response back to the browser client!
Following are minimum steps needed to create any servlet:

Write servlet
- Import the necessary Java packages.
- Inherit from GenericServlet or the HTTP convenience class HttpServlet.
- Override the service method or the doXXX() methods.
- Save the file with a .java filename extension.

2. Deploy the servlet in a container. The process for deploying (installing) a servlet into a webserver (container) varies from webserver to webserver.

Test the servlet by invoking the servlet from a JDK1.8-compatible browser. The URL typically will be of the form:
- http://host_name:port/servletcontext/servlet/servlet_class_name
- For example: http://localhost:9090/WildFlyServletsTestDemo/HelloWorld,

where WildFlyServletsTestDemo is the context root and HelloWorld is the name of the servlet.

If the server is installed on the same machine as the browser, then the server name can also be referred to as localhost.

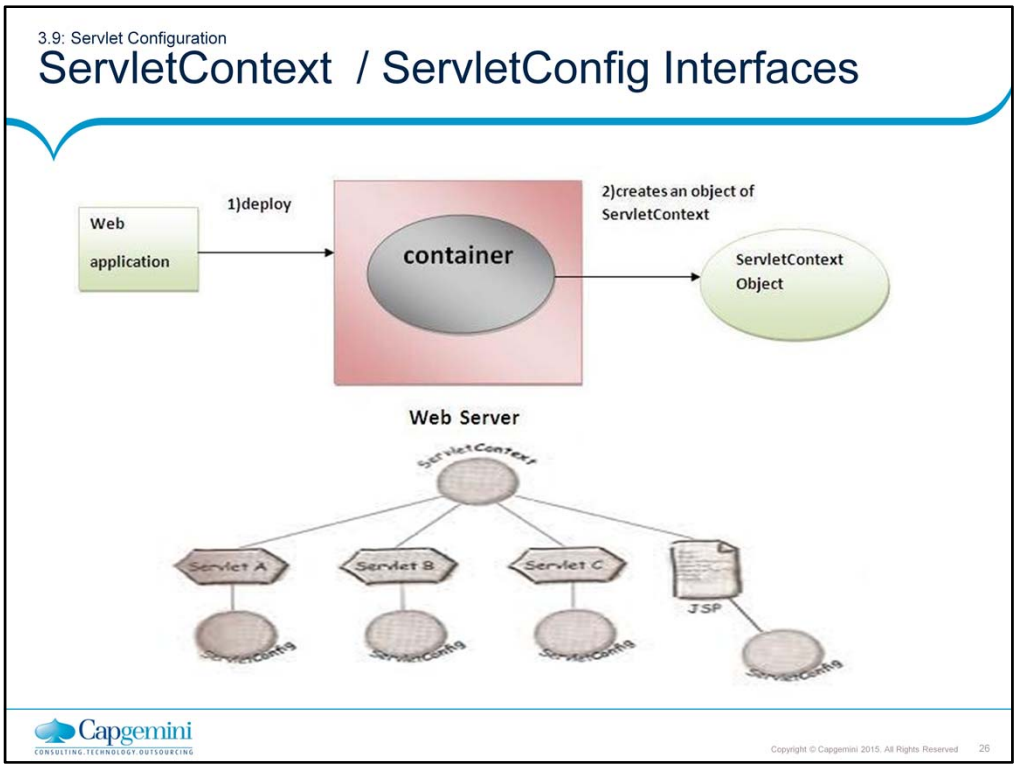
Note:
We shall use Eclipse Luna 4.4 with WildFly 8.x
To create a web application in Eclipse, follow the given steps:

In Project Explorer, right-click and select New → Dynamic Web Project.
Give a name for the project, say WildFlyServletsTestDemo. Ensure target runtime selected is WildFly 8.x, configure the server.
Check on the check box to create (web.xml – deployment descriptor) as it is optional under Servlets 3.0 and click Finish.
Create servlet class under src folder and within package controller . Refer HelloWorld.java class. At this point compile servlet, but since we are using IDE, this process is automatic. The servlet-api.jar is already part of build path, since we are using WildFly.
As we make use of annotation thus no changes required in web.xml.
Start the browser in Eclipse and run the servlet by invoking it with the following URL:

http://localhost:9090/WildFlyServletsTestDemo/HelloWorld

Hello World should be seen on the browser window!

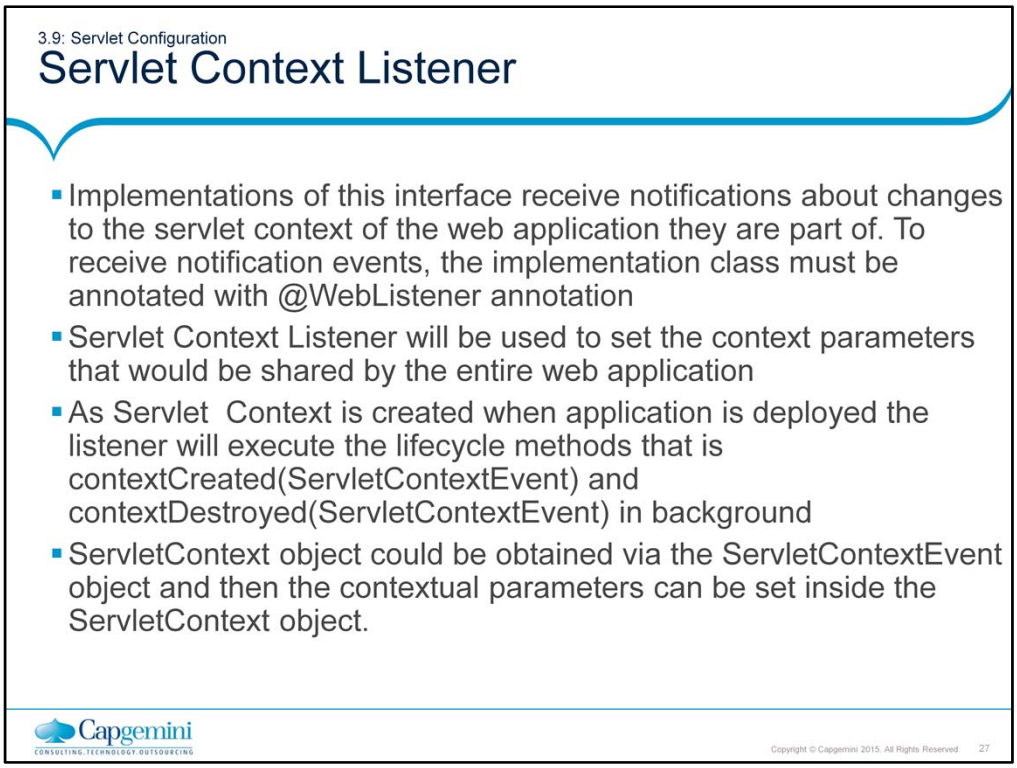
The web application could be packaged into a .WAR file.
Right click and select context root → Export → WAR File. Select the destination and click Finish.
This WAR file could be dropped into an web server like Wildfly and Tomcat (into its webapps folder) and invoke servlet using any browser!



The Container is responsible for creating the ServletContext object when the application gets deployed. ServletContext Object can be shared among multiple web components.

ServletConfig object will be created by the container for every servlet. This object cannot be shared between multiple web components.

More on these interfaces in the next subsequent slides



ServletContextListener is a listener which would get notified for the changes in the ServletContext once the ServletContext object gets created.

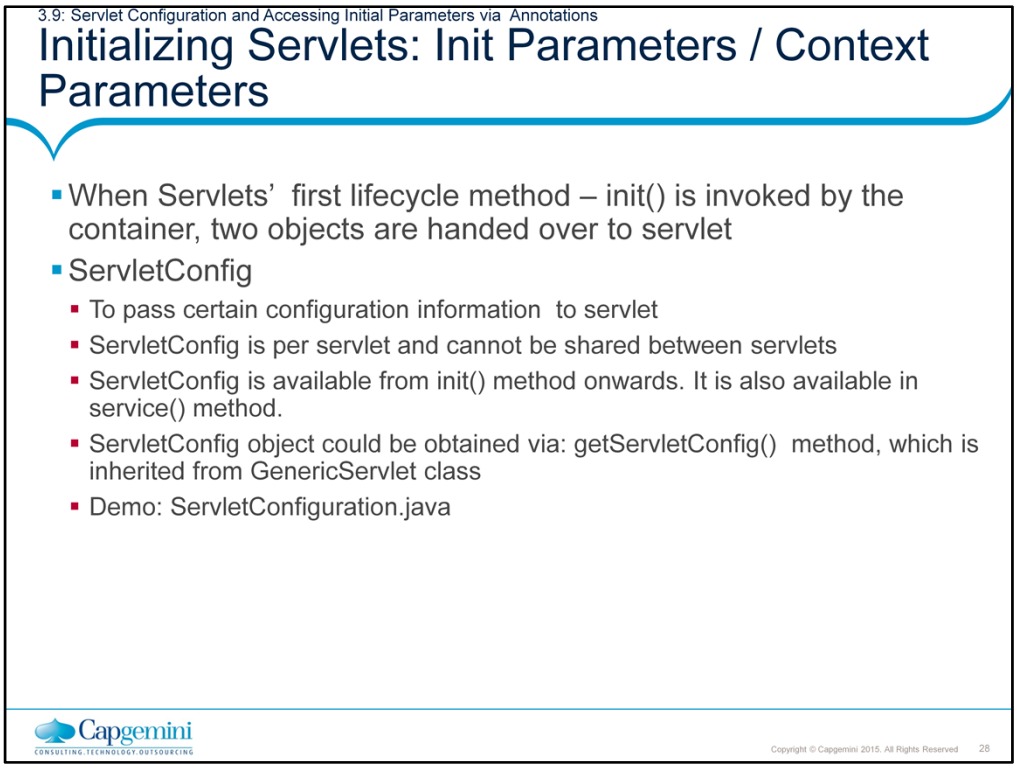
As ServletContext is created per Web application , data can be configured in listener which can be shared throughout the application.

Once the Listener is implemented need to override the life cycle methods as contextCreated() and contextDestroyed(), each of them taking a parameter of type ServletContextEvent; this object can be used to obtain the ServletContext object. (ServletContextEvent.getServletContext())

Refer below listing to have an idea about the Listener:

```
@WebListener
public class TestServletContextListener implements ServletContextListener {
public void contextDestroyed(ServletContextEvent sc) {
}
public void contextInitialized(ServletContextEvent sc) {
}

}
```



There are 2 versions of init:
init()
init(ServletConfig config)

Both these methods could be overridden. If we override parameterized version of init(ServletConfig config) there is a need to give a call explicitly to super.init(config). By seeing this call the container would invoke the parameterized init() from the GenericServlet hierarchy and then give a call for the default version of init() handing over the ServletConfig and ServletContext objects.

Even if we override default version of init(), the container would still invoke the parameterized init(ServletConfig config) from the GenericServlet hierarchy and then give a call for the default version of init() handing over the ServletConfig and ServletContext objects.

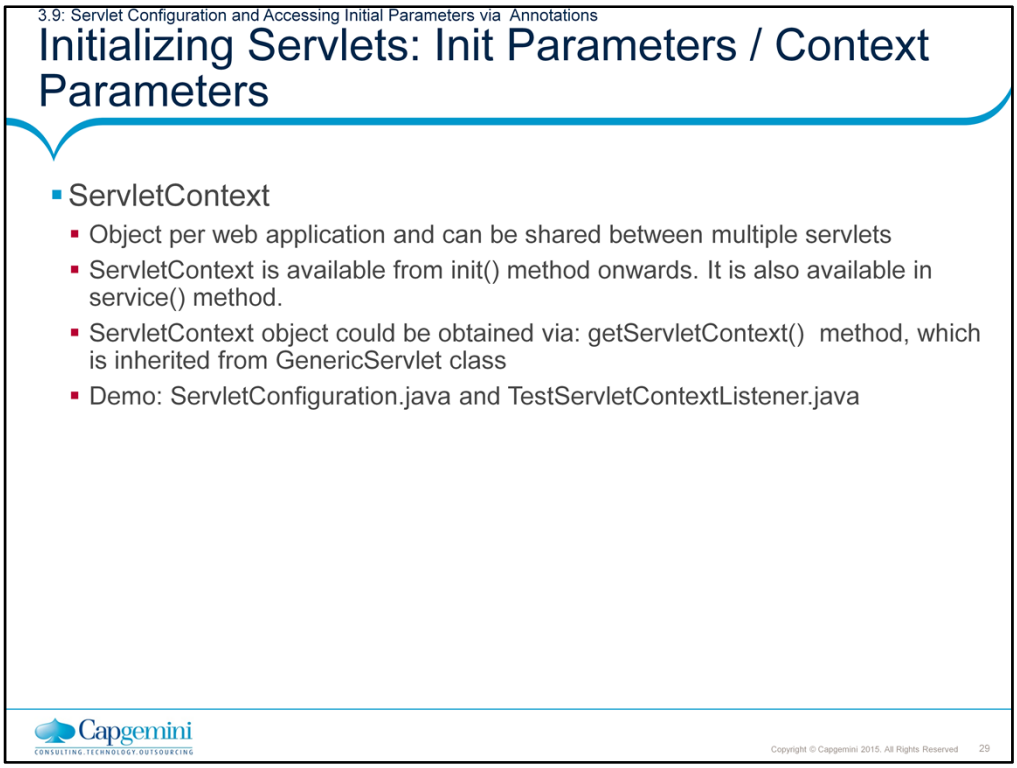
This is due to Servlet Hierarchy discussed in previous slides.

Since Servlet 3.0 we have made use of annotation to declare init parameter. Example:
@WebInitParam(name = "name", value = "My Servlet")

For Servlet Context parameters, there is a need to configure a Listener :
ServletContextListener – which will implement the lifecycle methods of ServletContext.
And then the contextual parameters could be set.

Refer Demo TestServletContextListener.java for same.

The context parameters could be then retrieved inside a servlet.



There are 2 versions of init:
init()
init(ServletConfig config)

Both these methods could be overridden. If we override parameterized version of init(ServletConfig config) there is a need to give a call explicitly to super.init(config). By seeing this call the container would invoke the parameterized init() from the GenericServlet hierarchy and then give a call for the default version of init() handing over the ServletConfig and ServletContext objects.

Even if we override default version of init(), the container would still invoke the parameterized init(ServletConfig config) from the GenericServlet hierarchy and then give a call for the default version of init() handing over the ServletConfig and ServletContext objects.

This is due to Servlet Hierarchy discussed in previous slides.

Since Servlet 3.0 we have made use of annotation to declare init parameter. Example:
@WebInitParam(name = "name", value = "My Servlet")

For Servlet Context parameters, there is a need to configure a Listener :
ServletContextListener – which will implement the lifecycle methods of ServletContext.
And then the contextual parameters could be set.

Refer Demo TestServletContextListener.java for same.

The context parameters could be then retrieved inside a servlet.

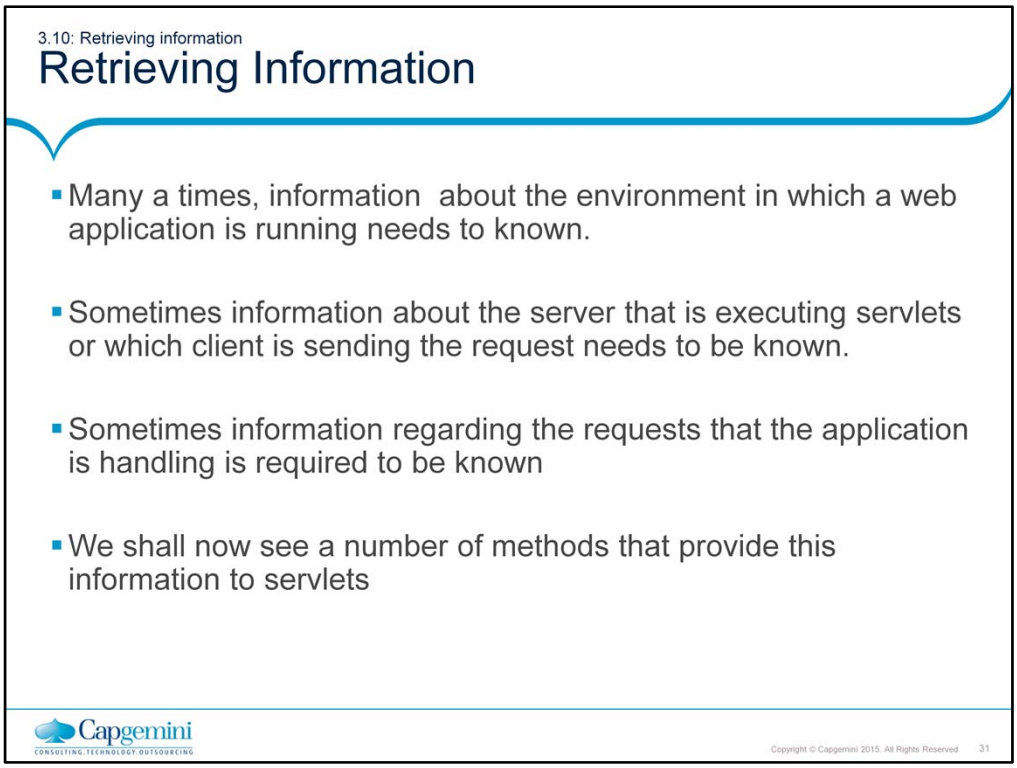
Note:
Refer com.igate.ch3.ServletConfiguration.java class. Notice that we have used GenericServlet to create the servlet.

For the ServletContext , a Listener is created called TestServletContextListener that would implement the Lifecycle methods of ServletContext.

```
public class InitSnoop extends GenericServlet {
  public void service(ServletRequest req, ServletResponse res)
          throws ServletException, IOException {
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();
    out.println("Init Parameters:");
    //retrieves all init parameters from web.xml
    Enumeration params = getInitParameterNames();
    while (params.hasMoreElements()) {
        String name = (String) params.nextElement();
        out.println(name + ": " + getInitParameter(name));
      }
   }
}
```

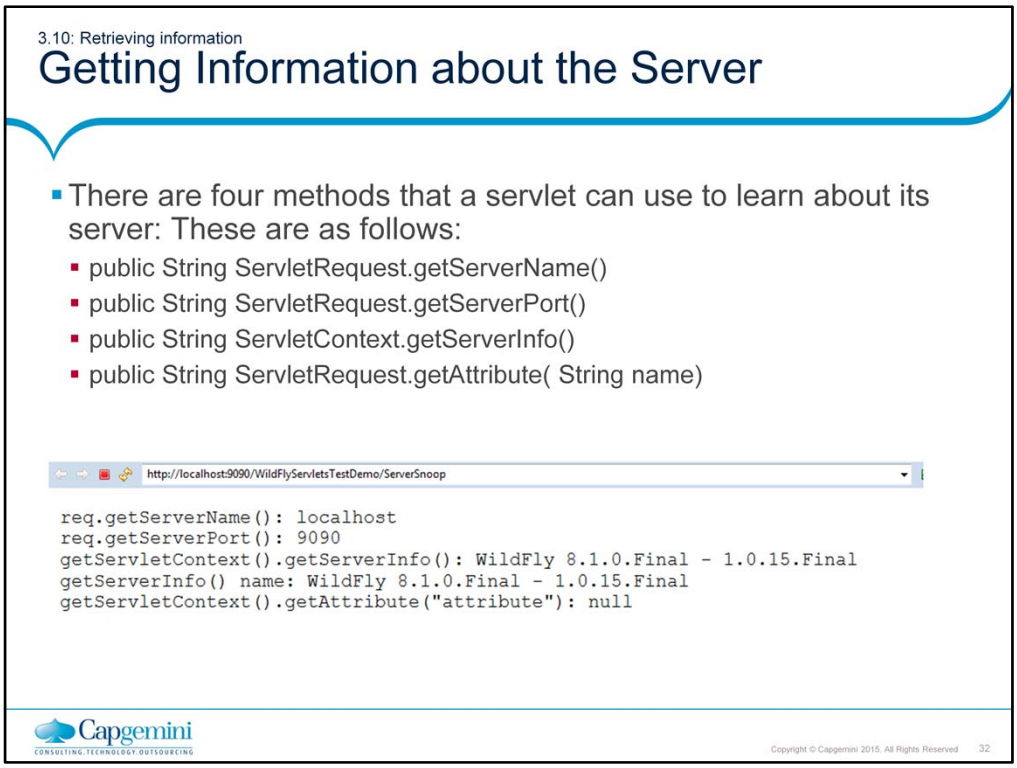
Notice how the getInitParameter() and getInitParameterNames() have been used to retrieve the appropriate init information. Invoke this servlet as: http://localhost:8080/Servlet-Demo/initsnoop

The browser shows the init parameters with values as shown above



In the Servlet Hierarchy, there is a Listener associated with different objects like ServletContext, HttpSession etc; that would be invoked when these objects are created. A Listener is a java class that would implement the ServletContextListener interface and would implement the lifecycle methods of ServletContext:
contextInitilized(ServletContextEvent se)
contextDestroyed(ServletContextEvent se)

We are using the annotation @WebListener above the Listener class name that would instruct the container to create the ServletContextListener and invoked it when a ServletContext object is created by Container, Data or configuration information that needs to be shared across multiple servlets could be set up in the ServletContext here. ServletContext is obtained via the ServletContextEvent.getServletContext() method.



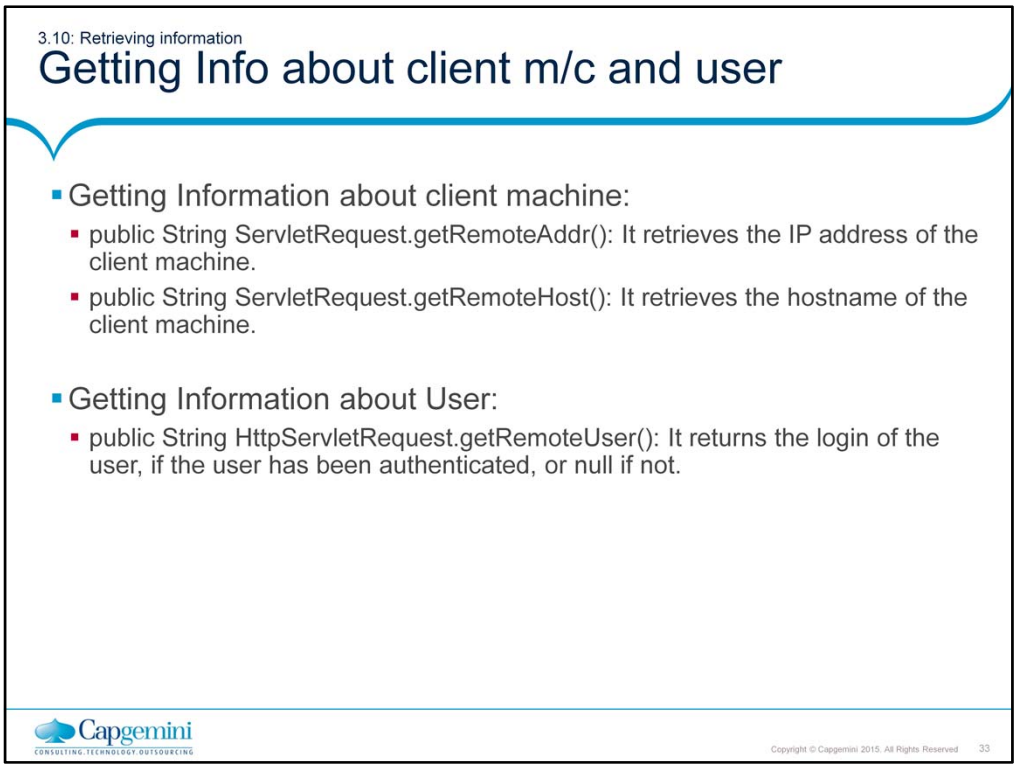
Retrieving Information:
Getting Information about the Server:
A servlet can find out much about the server in which it is executing. It can learn the hostname, listening port, and server software, among other things. A servlet can display this information to a client, use it to customize its behavior based on a particular server package, or even use it to explicitly restrict the machine on which the servlet will run.

getServerName() method returns the host name of the server to which the request was sent.

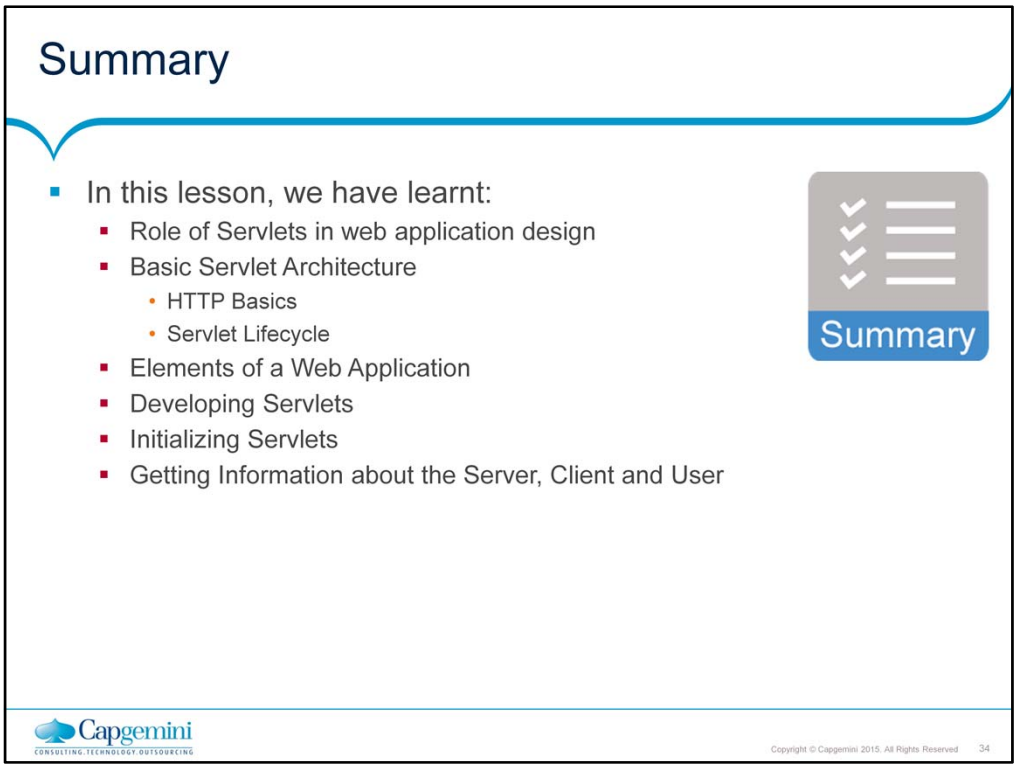
getServerPort() method returns the port number to which the request was sent

getServerInfo() method returns the name and version of the server software as: JBossWeb/2.0.1.GA.

getAttribute() returns the value of the named server attribute as an Object or null if the attribute does not exist. The attributes are server dependent. We shall see this in detail in the session on inter-servlet communication.



Retrieving Information:
Getting Info About client m/c and user:
A servlet has the ability to find out about the client machine and for pages requiring authentication, about the actual user. This information can be used for logging access data, associating information with individual users, or restricting access to certain clients.
A servlet can use getRemoteAddr() and getRemoteHost() to retrieve the IP address and hostname of the client machine, respectively.
The information comes from the socket that connects the server to the client, so the remote address and hostname may be that of a proxy server. An example remote address might be “192.26.80.1320” while an example of remote host might be “dist.engr.com”.
The method getRemoteUser () of the HttpServletRequest gives the username of the client. With the remote user’s name, a servlet can save information about each client. Over the long term, it can remember each individual’s preferences. For the short tern, it can remember the series of pages viewed by the client and use them to add a sense of state to a stateless HTTP protocol. A simple servlet that uses getRemoteUser() can greet its clients by name and remember when each user last logged in.



Summary:
This lesson introduced us to servlets, web applications. We have seen how to create simple servlets, install on server and execute it. We learnt about the servlet lifecycle and the Servlet API. We also saw how to initialize servlets using init() method and how to find information about the server, the client making the request and the user.

Review Questions

- Question 1: Which Method in a HttpServlet is not recommended to be overridden?
  - Option 1: init
  - Option 2: destroy
  - Option 3: service
  - Option 4: doGet
  - Option 5: doPost

- Question 2: What is the superclass of HttpServlet?
  - Option 1: GenericServlet
  - Option 2: Servlet
  - Option 3: SuperHttpServlet

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 35

Review Questions

- Question 3: Servlets are:
  - Option 1: Server-side components
  - Option 2: Client-side components
  - Option 3: Neither Client-side or Server-side components
  - Option 4: Data tier components

- Question 4: When a servlet receives an HTTP GET Request for the first time, which of these methods will be called? (Assume the servlet is not preloaded)
  - Option 1: init
  - Option 2: doPost
  - Option 3: processRequest
  - Option 4: doGet
  - Option 5: service

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 36

Review Questions

- Question 5: What is the significance of using annotation @WebServlet("/HelloWorld") above public class HelloWorld extends HttpServlet {}
  - Option 1: Instruct container to create Servlet Hello with URL Pattern “/HelloWorld”
  - Option 2: Instruct container to create Servlet HelloWorld with URL Pattern “/Hello”
  - Option 3: Instruct container to create Servlet HelloWorld with URL Pattern “/HelloWorld”
  - Option 4: No significanc

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved

Review Questions

- Question 6: How to pass Initialization parameters to servlet
  - Option 1: creating mapping in xml file
  - Option 2: making use of annotation @WebInitParam
  - Option 3: making use of annotation @WebContextParam
  - Option 4: configuring a listener

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved