(1)What is servlet? Explain a life cycle of a servlet with neat sketch?

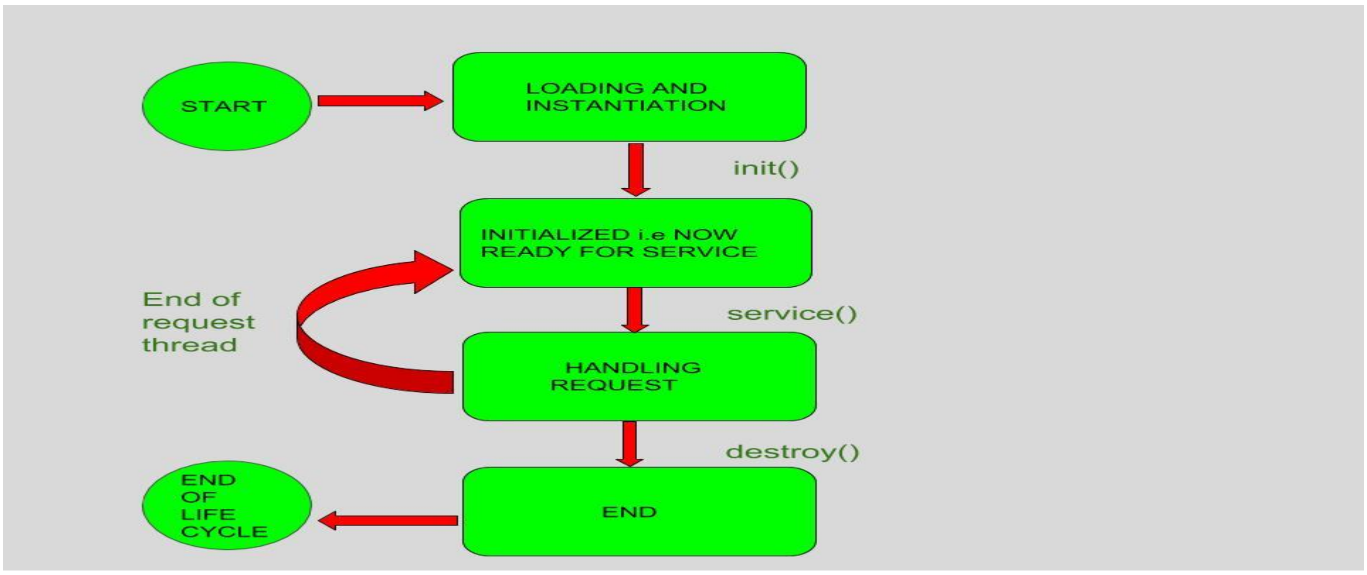
Servlets are server side programs that run on a web or application server and acts as a middle layer between requests coming from a web browser and databases or application on the server

Life Cycle of a Servlet

The entire life cycle of a Servlet is managed by the **Servlet container** which uses the **javax.servlet.Servlet** interface to understand the Servlet object and manage it.

Stages of the Servlet Life Cycle: The Servlet life cycle mainly goes through four stages,

- Loading a Servlet.
- Initializing the Servlet.
- Request handling.
- Destroying the Servlet.



1. Loading a Servlet:

The first stage of the Servlet lifecycle involves loading and initializing the Servlet by the Servlet container.

The Web container or Servlet Container can load the Servlet at either of the following two stages:

- Initializing the context, on configuring the Servlet with a zero or positive integer value.

- If the Servlet is not preceding stage, it may delay the loading process until the Web container determines that this Servlet is needed to service a request.

The Servlet container performs two operations in this stage:

Loading: Loads the Servlet class.

Instantiation: Creates an instance of the Servlet. To create a new instance of the Servlet, the container uses the no-argument constructor.

2. **Initializing a Servlet:**

After the Servlet is instantiated successfully, the Servlet container initializes the instantiated Servlet object.

The container initializes the Servlet object by invoking *Servlet.init (ServletConfig)* method which accepts *ServletConfig* object reference as parameter.

The Servlet container invokes the *Servlet.init (ServletConfig)* method only once, immediately after the *Servlet.init (ServletConfig)* object is instantiated successfully. This method is used to initialize the resources, such as JDBC data source.

Now, if the Servlet fails to initialize, then it informs the Servlet container by throwing the *ServletException* or *UnavailableException*.

3. Handling request:

After initialization, the Servlet instance is ready to serve the client requests. The Servlet container performs the following operations when the Servlet instance is located to service a request:

- It creates the *ServletRequest* and *ServletResponse* objects. In this case, if this is a HTTP request, then the Web container
  creates *HttpServletRequest* and *HttpServletResponse* objects which are subtypes of the *ServletRequest* and *ServletResponse* objects respectively.

- After creating the request and response objects it invokes the *Servlet.service (ServletRequest, ServletResponse)* method by passing the request and response objects.

4. Destroying a Servlet:

When a Servlet container decides to destroy the Servlet, it performs the following operations.

- It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.

- After currently running threads have completed their jobs, the Servlet container calls the destroy () method on the Servlet instance.

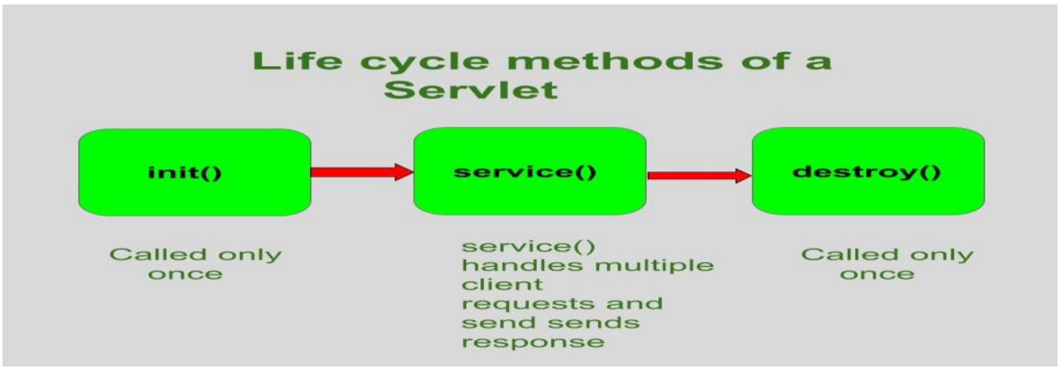
After the destroy() method is executed, the Servlet container releases all the references of this Servlet instance so that it becomes eligible for garbage collection.

(2)Explain life cycle methods of servlet in detail?

Servlet Life Cycle Methods

There are three life cycle methods of a Servlet :

1. init()
2. service()
3. destroy()


Life cycle methods of a Servlet
init()
service()
destroy()
Called only once
service() handles multiple client requests and send sends response
Called only once

1. init () method: The Servlet.init() method is called by the Servlet container to indicate that this Servlet instance is instantiated successfully and is about to put into service.

```
//init() method

public class MyServlet implements Servlet{

  public void init(ServletConfig config) throws ServletException {

    //initialization code
```

```
 }

//rest of code

}
```

2. service () method: The service() method of the Servlet is invoked to inform the Servlet about the client requests.

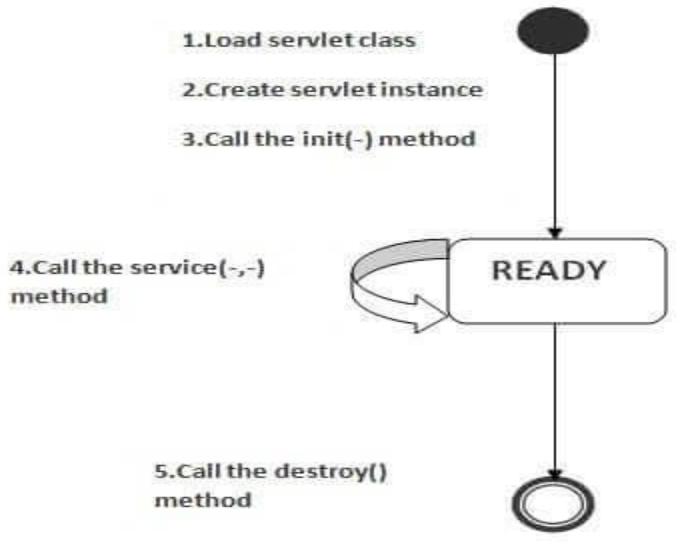
- This method uses ServletRequest object to collect the data requested by the client.
- This method uses ServletResponse object to generate the output content.

```
// service() method

public class MyServlet implements Servlet{

   public void service(ServletRequest req, ServletResponse res)

   throws ServletException, IOException {

         // request handling code

   }

   // rest of code

}
```

3. destroy () method: The destroy() method runs only once during the lifetime of a Servlet and signals the end of the Servlet instance.

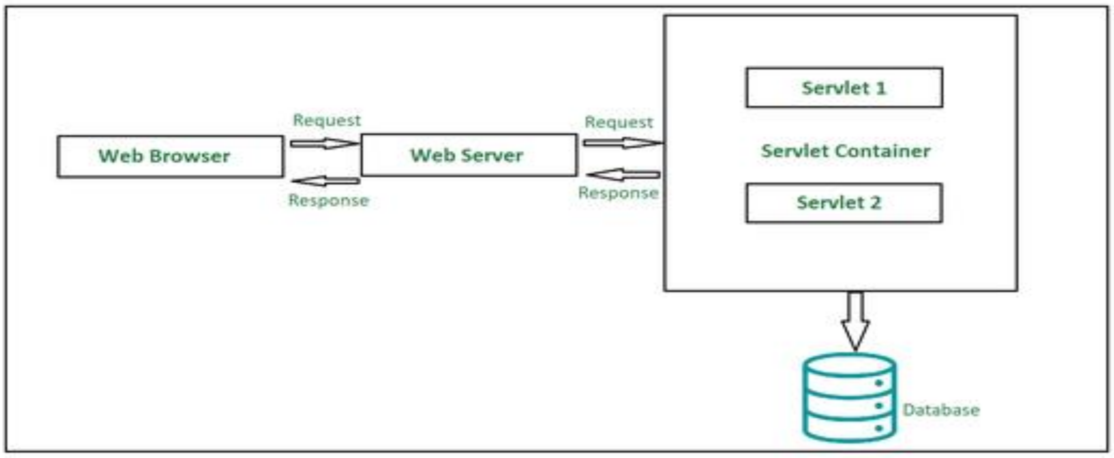
```
//destroy() method

public void destroy()
```

As soon as the destroy() method is activated, the Servlet container releases the Servlet instance.


1.Load servlet class
2.Create servlet instance
3.Call the init(-) method
4.Call the service(-,-) method
READY
5.Call the destroy() method

(3) Explain the servlet architecture with the help of neat diagram?

Components of Servlet Architecture
Below is the high level architecture diagram of servlet.
Servlet Architecture



1. Client
The client shown in the architecture above is primarily working as a medium who is sending out HTTP requests over to the web server and again processing the response it gets back from the server. As we can see in the diagram, our client here is the web browser.
2. Web Server
Primary job of a web server is to process the requests and responses that a user sends over time and maintain how a web user would be able to access the files that has been hosted over the server. The server we are talking about here is a software which manages access to a centralized resource or service in a network.. There are precisely two types of webservers:
1. **Static web server**
2. **Dynamic web server**

3. Web Container
Web container is another typical component in servlet architecture which is responsible for communicating with the servlets. Two prime tasks of a web container are:

- **Managing the servlet lifecycle**
- **URL mapping**

(4) Write a servlet program that displays “MVGR Autonomous” message on webpage

Assuming that you have already set up a Java development environment and a web server, here's an example of a simple servlet program that displays "MVGR Autonomous" message on a webpage:

```
java
Copy code
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MyServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    response.getWriter().println("<h1>MVGR Autonomous</h1>");
  }
}
```

In this program, we define a servlet named MyServlet that extends HttpServlet class. The doGet method of this servlet is called when a GET request is made to the servlet.

Within the doGet method, we set the content type of the response to "text/html" using the setContentType method of the HttpServletResponse object. Then we write the "MVGR Autonomous" message in an HTML heading level 1 (<h1>) using the getWriter method of the HttpServletResponse object.

To use this servlet, you need to compile the Java code and package it into a WAR file. Then, you can deploy the WAR file to your web server and access the servlet using its URL. For example, if you deploy the WAR file to the root context of your web server, you can access the servlet using the URL http://localhost:8080/MyServlet. This will display the "MVGR Autonomous" message on the webpage.

(5) Discuss various classes and interfaces of the javax.servlet package

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.

The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Interfaces in javax.servlet package

There are many interfaces in javax.servlet package. They are as follows:

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener
12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

Classes in javax.servlet package

There are many classes in javax.servlet package. They are as follows:

1. GenericServlet
2. ServletInputStream
3. ServletOutputStream
4. ServletRequestWrapper
5. ServletResponseWrapper
6. ServletRequestEvent
7. ServletContextEvent
8. ServletRequestAttributeEvent
9. ServletContextAttributeEvent
10. ServletException
11. UnavailableException

Interfaces in javax.servlet.http package

There are many interfaces in javax.servlet.http package. They are as follows:

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener
7. HttpSessionActivationListener
8. HttpSessionContext

Classes in javax.servlet.http package

There are many classes in javax.servlet.http package. They are as follows:

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils

6) Explain about Generic servlet class with an example.

GenericServlet class

1. **GenericServlet class**
2. **Methods of GenericServlet class**
3. **Example of GenericServlet class**

GenericServlet class
implements Servlet, ServletConfig and Serializable interfaces. It provides the implementation of all the methods of these interfaces except the service method.

GenericServlet class can handle any type of request so it is protocol-independent.

You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Methods of GenericServlet class

There are many methods in GenericServlet class. They are as follows:

1. public void init(ServletConfig config) is used to initialize the servlet.
2. public abstract void service(ServletRequest request, ServletResponse response) provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. public void destroy() is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. public ServletConfig getServletConfig() returns the object of ServletConfig.
5. public String getServletInfo() returns information about servlet such as writer, copyright, version etc.
6. public void init() it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. public ServletContext getServletContext() returns the object of ServletContext.
8. public String getInitParameter(String name) returns the parameter value for the given parameter name.
9. public Enumeration getInitParameterNames() returns all the parameters defined in the web.xml file.
10. public String getServletName() returns the name of the servlet object.
11. public void log(String msg) writes the given message in the servlet log file.
12. public void log(String msg,Throwable t) writes the explanatory message in the servlet log file and a stack trace.

Servlet Example by inheriting the GenericServlet class

Let's see the simple example of servlet by inheriting the GenericServlet class.

It will be better if you learn it after visiting steps to create a servlet.

File: First.java

```
import java.io.*;
import javax.servlet.*;
```

```
public class First extends GenericServlet{
public void service(ServletRequest req,ServletResponse res)
throws IOException,ServletException{
res.setContentType("text/html");
PrintWriter out=res.getWriter();
out.print("<html><body>");
out.print("<b>hello generic servlet</b>");
out.print("</body></html>");
}
}
```

GenericServlet class

```
import java.io.*;
import javax.servlet.*;

public class First extends GenericServlet{
public void service(ServletRequest req,ServletResponse res) throws
IOException,ServletException
{

res.setContentType("text/html");

PrintWriter out=res.getWriter();
out.print("<html><body>");
out.print("<b>hello generic servlet</b>");
out.print("</body></html>");

}
}
```

(7) Explain classes and interfaces supported by javax.servlet package

Types of Packages

There are two types of packages in Java Servlet that are providing various functioning features to servlet Applications. The two packages are as follows:

1. javax.servlet package
2. javax.servlet.http package

Type 1: javax.servlet package: This package of Servlet contains many servlet interfaces and classes which are capacity of handling any types of protocol sAnd This javax.servlet package containing large interfaces and classes that are invoked by the servlet or web server container as they are not specified with any protocol.
Type 2: javax.servlet.http package: This package of servlet contains more interfaces and classes which are capable of handling any specified http types of protocols on the servlet. This javax.servlet.http package containing many interfaces and classes that are used for http requests only for servlet

Interfaces and Classes in javax.servlet package

Interfaces in javax.servlet package

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener
12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

Classes in javax.servlet package

The Classes are in javax.servlet package are listed below:

1. GenericServlet
2. ServletInputStream
3. ServletOutputStream
4. ServletRequestWrapper
5. ServletResponseWrapper
6. ServletRequestEvent
7. ServletContextEvent
8. ServletRequestAttributeEvent
9. ServletContextAttributeEvent
10. ServletException
11. UnavailableException

Servlet: This interface describes and connects all the methods that a Servlet must implement. It includes many methods to initialize the destroy of the Servlet, and a general (service()) method which is handling all the requests are made to it. This Servlet interface is used to creating this servlet class as this class having featuring to

implementing these interfaces either directly or indirectly to within it on to fetching servlets
ServletRequest: This ServletRequest interface in which examining the methods for all objects as encapsulating data information about its all requests i.e. made to the servers, this object of the ServletRequest interface is used to retrieve the information data from the user
ServletResponse: An interface examining the methods for all objects which are returning their allowed responses from the servers and object of this current interfacing objects is used to estimate the response to the end-user on the system
ServletConfig: declaring this interface ServletConfig useful to gaining accessing the configuration of its main parameters which are passing through the Servlets during the phase time of initialization and this ServletConfig object is used for providing the information data to the servlet classes external to explicitly.
ServletContext: The object of the ServletContext interface is very helpful to featuring the info. data to the web applications are explaining to it for servlets
GenericServlet: This is a generic classes examination to implement the Servlet. if you want to write the Servlet's protocols other than the HTTP, then the easy way of doing this is to extend GenericServlet rather than by directly implementing the Servlet interfaces
ServletException: it is an exception that can be thrown when the Servlet invoking a problem of some examples
ServletInputStream: This class ServletInputStream is used to reading the binary data from end user request
ServletContextEvent: in this any changes are made in the servlet context of its web application, this class notifies it to the end-user.
ServletOutputStream: This class ServletOutputStream is useful to send the transferring binary data to the end-user side of the system

Interfaces And Classes in javax.servlet.http package

Interfaces: The javax.servlet.http packages have provides these feature classes that are unique to handling these HTTP requests allowing from it. It provides the HttpServlet classes that is usable as it accesses the selectively interfaces from javax.servlet class.

Interfaces in javax.servlet.http

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener
7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)
9. HttpServletRequest – as the extension to ServletRequest interface is using for features specified to HTTP

10. HttpServletResponse – as this extension to ServletResponse interface is using for functions are similar to HTTP
11. HttpSession – this interface featuring the accessing to the sessions of tracking for API
12. HttpSessionAttributeListener – This interface notifies if any changes/edits are prefetched in this HttpSession attribute
13. HttpSessionListener – This HttpSessionListener interface notified any changes/edits are prefetched in this interface HttpSession lifecycle span process

Classes in javax.servlet.http package

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

HttpServlet: in this HttpServlet purely abstracted class having features as functionality to extending and applying on the HTTP requests. They have like Service() method that is declared in the Servlet interfaces will now call its methods similar to doGet() and the doPost(), which are enabled to providing behavior to the Calling Servlet
Cookie: This Class provides the feature Servlet an interface for the storage of small portions of data information on the end-user computer or system.
HttpServletRequestWrapper and HttpServletResponseWrapper: this two wrapper classes allowing capability of the HttpServletResponse and HttpServletRequest interfaces to the servlet by its functions
HttpSessionEvent: This class HttpSessionEvent notified as any activity or changes/editing are encountered in the session of web applications in servlet
HttpSessionBindingEvent: This class notified when any attribute is bounded, unbounded or replaced in any Current session

8) Explain about HttpServletRequest and HttpServletResponse interfaces

The `HttpServletRequest` and `HttpServletResponse` interfaces are part of the Java Servlet API and provide methods for handling HTTP requests and generating HTTP responses, respectively.

The `HttpServletRequest` interface extends the `ServletRequest` interface and provides additional methods for handling HTTP-specific request information.

Some of the methods provided by `HttpServletRequest` are:

- `getMethod()`: Returns the HTTP method of the request (e.g., GET, POST, PUT, DELETE).
- `getRequestURI()`: Returns the URI of the request.
- `getParameter(String name)`: Returns the value of a request parameter with the given name.
- `getSession()`: Returns the session associated with the request, or creates a new session if one does not exist.

- ***HTTPServletRequest*** Extends the ServletRequest interface to provide request information for HTTP servlets.
- The servlet container creates an HttpServletRequest object and passes it as an argument to the servlet's service methods (doGet, doPost, etc).
- It Contains all the client's request information.
- The HttpServletRequest breaks a request down into parsed elements, such as request URI, query arguments and headers.
- Various get methods allow you to access different parts of the request.
- **requestURI –** URL sent by browser
- **Parameters -** The HttpServletRequest provides methods for accessing parameters of a request. The methods getParameter(), getParameterValues() and getParameterNames() are offered as ways to access the arguments.
- **Attributes –** The request object defines a method called getAttribute(). The servlet interface provides this as a way to include extra information about the request that is not covered by any of the other HttpServletRequest methods.
- **ServletInputStream –** The ServletInputStream is an InputStream that allows your servlets to read all of the request's input following the headers.
- ***HTTPServletResponse*** Extends the ServletResponse interface and can perform these tasks.
- **Set Response Codes –**
  - The response code for a request is a numeric value that represents the status of the response. For example, 200 represents a successful response, 404 represents a file not found.
- **Set Headers –**

- Headers for the response can be set by calling setHeader, specifying the name and value of the header to be set.

- **Send Redirects –**
  - The sendRedirect method is used to issue a redirect to the browser, causing the browser to issue a request to the specified URL.
  - The URL passed to sendRedirect must be an absolute URL—it must include protocol, machine, full path, and so on.
- **Set ServletOutputStream –**
  - The ServletOutputStream is obtained by calling getOutputStream on the HttpServletResponse.
  - It is a subclass of OutputStream that contains a number of convenient print and println methods.
  - Data written to the ServletOutputStream goes straight back to the browser.

9) Write a servlet to read the parameters (First Name, Middle Name, Last Name, age, gender) from the user. if the candidate age >18 display `ELIGIBLE` else `NOT ELIGIBLE` and also display the collected details?

(10) Explain how servlet can read initialization parameters with an example.

Reading Form Data using Servlet

- **Servlets handles form data parsing automatically using the following methods depending on the situation:**
  - **getParameter(): You call request.getParameter() method to get the value of a form parameter.**
  - **getParameterValues(): Call this method if the parameter appears more than once and returns multiple values, for example checkbox.**
  - **getParameterNames(): Call this method if you want a complete list of all parameters in the current request.**

Reading Initialization Parameters

- **Using Servlet Config:**
- **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.**
- **If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.**
- **Methods**
  - **getInitParameter(String name): returns a String value initialized parameter**
  - **getInitParameterNames(): returns the names of the servlet's initialization parameters as an Enumeration of String objects**
  - **getServletContext(): returns a reference to the ServletContext**
  - **getServletName(): returns the name of the servlet instance**

Syntax to provide the initialization parameter for a servlet

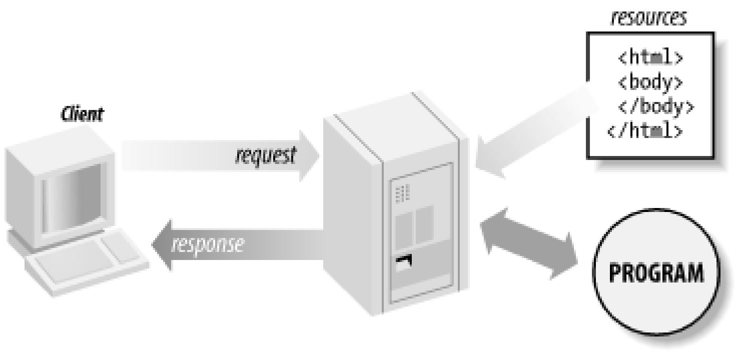
The init-param sub-element of servlet is used to specify the initialization parameter for a servlet.

```
<web-app>

<servlet>

......

<init-param>

<param-name>driver</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</init-param>

......

</servlet>

</web-app>
```

Retrieve ServletConfig

```
    ServletConfig sc = getServletConfig();

    out.println(sc.getInitParameter(“driver"));
```

11) Write about Http request/Response Model in detail with neat sketch.



The HTTP (Hypertext Transfer Protocol) request/response model is the basis of communication between clients (such as web browsers) and servers on the World Wide Web. In this model, clients send HTTP requests to servers, and servers respond with HTTP responses that contain the requested content or an error message.

The HTTP request/response model can be represented by the following diagram:

luaCopy code

```
CLIENT SERVER | | | HTTP Request| |------------->| | | | HTTP Response |<------------| | |
```

Here's a breakdown of each step in the HTTP request/response model:

1. The client sends an HTTP request to the server. The request consists of a request line, headers, and an optional message body. The request line specifies the method (such as GET, POST, or PUT), the URI (Uniform Resource Identifier) of the requested resource, and the version of the HTTP protocol being used. The headers provide additional information about the request, such as the user agent, the content type, and the content length. The message body contains any data associated with the request, such as form data or file uploads.
2. The server receives the HTTP request and processes it. The server determines the appropriate action to take based on the request method and URI. For

example, if the method is GET and the URI is "/index.html", the server might retrieve the file "index.html" from the file system and return it in the response.

3. The server sends an HTTP response back to the client. The response consists of a status line, headers, and an optional message body. The status line specifies the HTTP version, a status code (such as 200 for success or 404 for not found), and a status message (such as "OK" or "Not Found"). The headers provide additional information about the response, such as the content type, the content length, and any cookies that need to be set. The message body contains the requested content or an error message.
4. The client receives the HTTP response and processes it. If the status code indicates success (such as 200), the client typically displays the content in the web browser. If the status code indicates an error (such as 404), the client might display an error message or perform some other appropriate action.

In summary, the HTTP request/response model is a simple and powerful way for clients and servers to communicate on the web. By following this model, web developers can create dynamic and interactive web applications that can be accessed by users around the world.

The parameters are the way in which a client or user can send information to the Http Server.

The HTTPServletRequest interface includes methods that allow you to read the names and values of parameters that are included in a client request.

The HttpServletResponse Interface provides functionality for sending response to client.

The browser uses two methods to pass this information to web server. These methods are GET Method and POST Method.

Describe how an Http servlet handles its client requests and responses

An HTTP servlet is a Java class that runs on a web server and handles client requests and responses by implementing the HttpServlet interface provided by the Java Servlet API. The servlet handles the requests and responses by following a well-defined lifecycle that includes initialization, request handling, and destruction.

Here's a step-by-step description of how an HTTP servlet handles client requests and responses:

1. Initialization: When the servlet is first loaded, the web server calls its `init()` method to perform any necessary initialization. The `init()` method is typically

used to set up the servlet's environment, such as loading configuration parameters and initializing database connections.

2. Request Handling: When a client sends an HTTP request to the web server, the web server forwards the request to the appropriate servlet. The servlet then calls its `service()` method to handle the request. The `service()` method determines the type of HTTP request (such as GET, POST, or PUT) and dispatches the request to the appropriate method (`doGet()`, `doPost()`, etc.). The servlet then generates an HTTP response by writing the response data to the response output stream. The `service()` method then sends the response back to the web server, which in turn sends it back to the client.
3. Request processing: The servlet extracts the relevant information from the HTTP request, such as headers, query parameters, or form data, and performs any necessary processing. This can include database queries, business logic, or any other custom logic required to generate the response.
4. Response generation: The servlet generates the HTTP response by writing data to the response output stream. This can include HTML content, JSON data, images, or any other type of data that can be represented as text. The servlet can also set response headers (such as the content type and encoding) and set cookies to be sent back to the client.
5. Destruction: When the servlet is no longer needed (for example, when the web server is shut down or when the servlet is replaced with a new version), the web server calls its `destroy()` method to perform any necessary cleanup.

The servlet's response is generated by writing data to the response output stream. This can include HTML content, JSON data, images, or any other type of data that can be represented as text. The servlet can also set response headers (such as the content type and encoding) and set cookies to be sent back to the client.

In summary, an HTTP servlet handles client requests and responses by following a well-defined lifecycle that includes initialization, request handling, and destruction. The servlet generates HTTP responses by extracting relevant information from the HTTP request, performing any necessary processing, and writing data to the response output stream. The servlet also sets response headers and cookies as required.

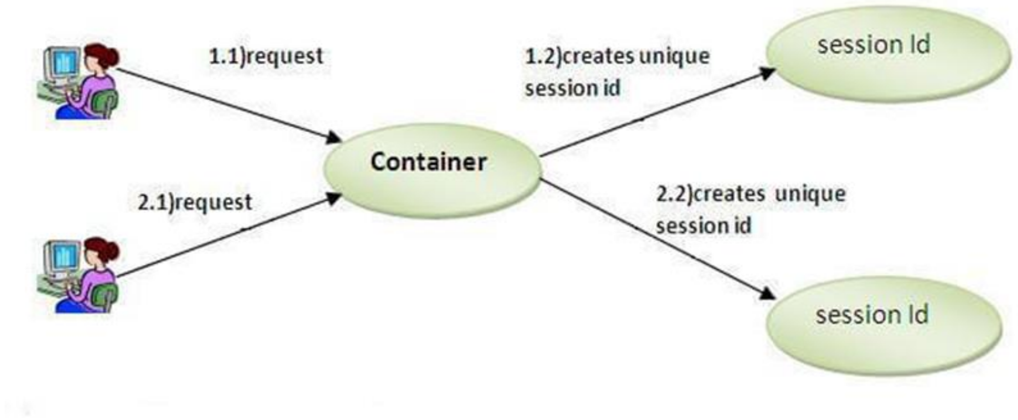
What are the different methods involved in the process of session management in servlets?

Session management in Servlets involves managing the session state and data of a user's interaction with a web application. Some of the common methods involved in session management in Servlets are:

1. **HttpSession interface: The HttpSession interface provides a way to store and retrieve session data on the server-side. It allows the server to associate a unique identifier with each client session and store session data using key-value pairs. The data stored in the session can be accessed and manipulated by multiple servlets or JSPs that are part of the same web application.**
2. **Cookies: Cookies are small pieces of data that are sent from the server to the client's browser and stored on the client-side. They can be used to maintain session state between requests. When a client sends a request to the server, it sends the cookie data along with it, allowing the server to retrieve the session data and continue the session.**
3. **URL rewriting: URL rewriting is a technique that involves adding session information to the URL of a web page. This information can be used by the server to maintain the session state between requests. The technique involves appending a session ID or a unique identifier to the URL of each page in the web application.**
4. **Hidden form fields: Hidden form fields are input fields that are not visible to the user but can be used to store session data. The server can include a hidden form field in an HTML form and set its value to the session ID. When the form is submitted, the session ID is sent to the server, allowing it to retrieve the session data and continue the session.**
5. **ServletContext: The ServletContext provides a way to store and retrieve application-level data that is accessible across multiple servlets or JSPs. The ServletContext can be used to store session data for all users of the web application. However, this approach is less efficient than using the HttpSession interface as it does not provide a unique session ID for each client.**

14) Explain session tracking using HttpSession.

- HttpSession Interface provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.
- Web container creates a session id for each user. The container uses this id to identify the particular user.
- The servlet container uses this interface to create a session between an HTTP client and an HTTP server.
- The session persists for a specified time period, across more than one connection or page request from the user.



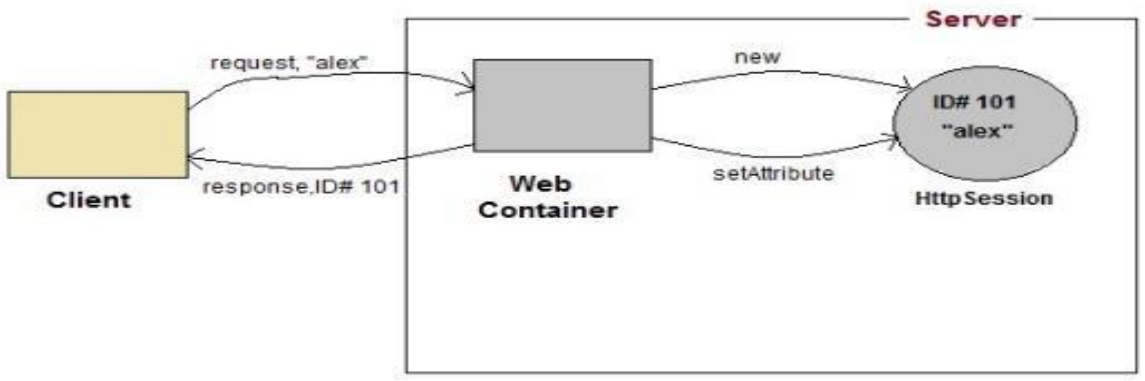
- **Get the HttpSession object: The HttpServletRequest interface provides two methods to get the object of HttpSession:**
  - **Public HttpSession getSession():Returns the current session associated with this request, or if the request does not have a session, creates one.**
  - **Public HttpSession getSession(boolean create):Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.**
- **Destroy Session**
  - **session.invalidate();**
- **Set/Get data in session**
  - **session.setAttribute(name,value);**
  - **session.getAttribute(name);**
- **Methods**
  - **public String getId():Returns a string containing the unique identifier value.**
  - **public long getCreationTime():Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.**
  - **public long getLastAccessedTime():Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.**
  - **public void invalidate():Invalidates this session then unbinds any objects bound to it.**

Steps to create session

On client's first request, the Web Container generates a unique session ID and gives it back to the client with response. This is a temporary session created by web container.

The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from.

The Web Container uses this ID, finds the matching session with the ID and associates the session with the request



15) Discuss security issues in servlets

Servlets, like any other web technology, can be vulnerable to security threats. Here are some of the most common security issues that can arise when developing servlet-based web applications:

1. Injection attacks: Injection attacks, such as SQL injection and cross-site scripting (XSS), are a common type of security issue in web applications. These attacks exploit vulnerabilities in the input validation mechanisms of the application to inject malicious code or data into the server.
2. Broken authentication and session management: Authentication and session management are critical components of web application security. If these components are not implemented correctly, attackers can exploit them to gain unauthorized access to the application or its data.
3. Cross-site request forgery (CSRF): CSRF attacks occur when an attacker tricks a user into performing an action on the application without their knowledge or consent. These attacks can be used to steal sensitive data, such as login credentials or personal information.
4. Insecure direct object references: Insecure direct object references occur when an application exposes sensitive data or resources, such as files or databases, to the client without proper access controls.
5. Insufficient logging and monitoring: Logging and monitoring are essential for detecting and responding to security incidents. If an application does not log

or monitor critical events, such as failed login attempts or access to sensitive data, it can be difficult to detect and respond to security threats.

To mitigate these security issues, developers should follow best practices for secure coding, such as validating user input, using secure authentication and session management mechanisms, implementing access controls, and logging and monitoring critical events. It is also essential to stay up-to-date with security patches and updates for the underlying technology stack used by the application. Additionally, conducting regular security audits and penetration testing can help identify and address security vulnerabilities in the application.

16) Explain any 3 Security issues in a Servlet environment.

Sure, here are three security issues in a Servlet environment:

1. Cross-site scripting (XSS): XSS is a type of injection attack where an attacker injects malicious code into a web page viewed by other users. This can be done by exploiting vulnerabilities in the input validation mechanisms of the application, such as allowing users to enter scripts in input fields. The injected code can then be used to steal user data, perform unauthorized actions, or spread malware. To prevent XSS attacks, developers should use input validation and output encoding techniques to ensure that user input is properly sanitized and that any output displayed to the user does not contain any malicious code.
2. Cross-site request forgery (CSRF): CSRF attacks occur when an attacker tricks a user into performing an action on a web application without their knowledge or consent. This is done by exploiting the user's existing session with the application. CSRF attacks can be used to steal user data, perform unauthorized actions, or modify sensitive data. To prevent CSRF attacks, developers should use anti-CSRF tokens that are unique to each user's session and are included in all forms and URLs that modify data in the application.
3. Injection attacks: Injection attacks are a type of vulnerability where attackers can inject malicious code or data into a web application by exploiting vulnerabilities in the input validation mechanisms of the application. Common types of injection attacks include SQL injection, where attackers inject malicious SQL queries into the application's database, and command injection, where attackers inject malicious commands into the application's shell. To prevent injection attacks, developers should use parameterized queries and prepared statements when accessing databases and should validate all user input to ensure that it does not contain any malicious code or data.