Advanced Java - Servlet Viva Questions & Answers

Q1. What is a Servlet?

Answer:

A **Servlet** is a server-side Java program that runs inside a **Servlet Container (Web Container)** and processes client requests (usually HTTP requests) to generate dynamic web content.

Servlets are mainly used to:

- Process client requests
- Interact with databases
- Generate dynamic HTML pages
- Manage sessions
- Implement business logic

A servlet executes on the **server**, not on the client browser.

Package:

```
javax.servlet.*
```

or (Tomcat 10+)

```
jakarta.servlet.*
```

Advantages

- Platform independent
- High performance
- Secure
- Multithreaded
- Portable

Follow-up Questions

- Where does a servlet execute?
- Which package contains Servlet classes?
- Is Servlet client-side or server-side?

Q2. Why do we use Servlets?

Answer

Servlets overcome the limitations of CGI (Common Gateway Interface).

Advantages include:

- Better performance
- Multithreading
- Platform independence
- Secure execution
- Easy database connectivity
- Session management support

Servlets can communicate with

- HTML
- JSP
- Database
- Spring Framework

Q3. What are the applications of Servlets?

Answer

Servlets are widely used in

- Login applications
- Student Management System
- Online Banking
- E-Commerce Websites
- Hospital Management
- Employee Management
- Library Management
- College ERP

Q4. What is a Servlet Container?

Answer

A Servlet Container (also called Web Container) is software that manages Servlets.

Examples

- Apache Tomcat
- GlassFish
- WildFly
- Jetty

Responsibilities

- Loads Servlets
- Calls lifecycle methods
- Creates request and response objects
- Session management
- Security
- Thread management

Q5. Explain Servlet Architecture.

Answer

Architecture

```
Browser
     |
HTTP Request
     |
Apache Tomcat
     |
Servlet Container
     |
Servlet
     |
Database
     |
Servlet
     |
HTTP Response
     |
Browser
```

The browser sends an HTTP request.

The Servlet Container receives it.

The servlet processes it.

A response is sent back to the browser.

Q6. Explain the Servlet Life Cycle.

Answer

Servlet Life Cycle consists of three important methods

1. init()

Executed only once.

Used for

- Initialization
- Database connection
- Reading configuration

2. service()

Executed for every request.

Calls

- doGet()
- doPost()

depending upon request type.

3. destroy()

Executed once before removing servlet.

Used for

- Closing database connection
- Releasing resources

Q7. What is init() method?

Answer

Syntax

```
public void init() throws ServletException
```

Purpose

- Initialize servlet
- Allocate resources

- Read initialization parameters

Called only once.

Q8. What is service() method?

Answer

Syntax

```
public void service(
ServletRequest req,
ServletResponse res)
```

Executed for every request.

If HttpServlet is extended

service()

automatically calls

`doGet()`

`or`

`doPost()`

depending upon HTTP request.

Q9. What is destroy()?

Answer

Syntax

```
public void destroy()
```

Executed only once.

Used to

- Release memory
- Close files
- Close database connection

Q10. Can init() execute multiple times?

Answer

No.

init()

executes only once when servlet is loaded.

service()

executes multiple times.

destroy()

executes once.

Q11. Difference between init() and service()

init()	service()
Executes once	Executes every request
Initialization	Processes request
Creates resources	Uses resources
Called by container	Called repeatedly

Q12. Difference between service() and doGet()

Answer

service()

receives every HTTP request.

Internally

service()

calls

`doGet()`

`or`

`doPost()`

depending upon request type.

Q13. What are the Types of Servlets?

Answer

Two types

GenericServlet

- Protocol independent

HttpServlet

- HTTP specific
- Most commonly used

Q14. Difference between GenericServlet and HttpServlet

GenericServlet	HttpServlet
Protocol independent	HTTP only
Overrides service()	Overrides doGet(), doPost()
Rarely used	Mostly used

Q15. Why is HttpServlet preferred?

Answer

HttpServlet supports

- GET
- POST
- PUT
- DELETE

- HEAD
- OPTIONS

Therefore it is suitable for web applications.

Q16. What is Deployment Descriptor?

Answer

Deployment Descriptor is

`web.xml`

It contains

- Servlet name
- Servlet class
- URL mapping
- Initialization parameters
- Welcome file
- Error pages

Q17. Where is web.xml stored?

Answer

`WEB-INF/web.xml`

Example

```
MyProject

WebContent

WEB-INF

web.xml
```

Q18. Why do we use web.xml?

Answer

It provides

- Servlet mapping
- Initialization parameters
- Session timeout
- Error page configuration
- Welcome page

Without annotations

Servlet configuration is done using

web.xml

Q19. What is URL Mapping?

Answer

URL Mapping connects

Browser URL

to

Servlet class.

Example

```
StudentServlet
```

```
↓
```

```
/student
```

Browser

```
http://localhost:8080/App/student
```

calls StudentServlet.

Q20. What is @WebServlet annotation?

Answer

Instead of using

```
web.xml
```

we can configure servlet using

```
@WebServlet("/student")
public class StudentServlet extends HttpServlet
{
}
```

Advantages

- No XML
- Easy configuration
- Less coding

Q21. What is ServletConfig?

Answer

ServletConfig is an interface used to pass **initialization parameters** to a specific servlet.

Every servlet has **its own ServletConfig object** created by the Servlet Container.

Package

```
import jakarta.servlet.ServletConfig;
```

(or `javax.servlet.ServletConfig` in older versions)

Uses

- Read initialization parameters
- Access servlet name
- Get ServletContext object

Example

```
String username = config.getInitParameter("username");
```

Advantages

- Easy configuration
- No hard coding
- Different values for different servlets

Q22. Who creates the ServletConfig object?

Answer

The **Servlet Container (Tomcat)** automatically creates the ServletConfig object during servlet initialization and passes it to the servlet.

Programmer **does not create** it manually.

Q23. How do you initialize ServletConfig?

Answer

Using **web.xml**

```
<servlet>
    <servlet-name>Student</servlet-name>

    <servlet-class>StudentServlet</servlet-class>

    <init-param>
        <param-name>college</param-name>
        <param-value>LBRCE</param-value>
    </init-param>

</servlet>
```

Retrieve value

```
String s=config.getInitParameter("college");
```

Q24. What are the important methods of ServletConfig?

Answer

Method	Purpose
getInitParameter()	Reads parameter
getInitParameterNames()	Returns all parameters
getServletName()	Returns servlet name
getServletContext()	Returns ServletContext

Q25. What is ServletContext?

Answer

ServletContext is an interface used to share information among **all servlets** of one web application.

Only **one ServletContext object** exists for one application.

Example

```
ServletContext ctx=getServletContext();
```

Q26. Difference between ServletConfig and ServletContext

ServletConfig	ServletContext
One per servlet	One per application
Servlet specific	Shared by all servlets
Stores init parameters	Stores global parameters
Cannot share data	Can share data

Q27. Why do we use ServletContext?

Answer

ServletContext is used

- Sharing data among servlets
- Reading global parameters
- Accessing resources
- Logging messages

Q28. What are Context Parameters?

Answer

Context Parameters are global parameters available to every servlet.

Configured inside

```
<context-param>

<param-name>college</param-name>

<param-value>LBRCE</param-value>
```

```
</context-param>
```

Access

```
ServletContext sc=getServletContext();

sc.getInitParameter("college");
```

Q29. What are Servlet Attributes?

Answer

Attributes are objects stored inside

- Request
- Session
- ServletContext

using

```
setAttribute()
```

Retrieve using

```
getAttribute()
```

Q30. Difference between Parameter and Attribute

Parameter	Attribute
Read-only	Read and Write
Configuration data	Runtime data
String only	Any Java Object
web.xml	Programmatically

Q31. What are the methods of ServletContext?

Answer

Important methods

```
getAttribute()

setAttribute()

removeAttribute()

getInitParameter()

log()

getRealPath()
```

Q32. What is RequestDispatcher?

Answer

RequestDispatcher is an interface used for communication between two resources.

It performs

- forward()
- include()

Example

```
RequestDispatcher rd=request.getRequestDispatcher("abc");
```

Q33. What is forward()?

Answer

forward()

transfers the request from one resource to another.

Example

```
rd.forward(request,response);
```

Browser URL

does NOT change.

Q34. What is include()?

Answer

include()

includes the output of another servlet/JSP.

Example

```
rd.include(request,response);
```

Useful for

- Header
- Footer
- Menu
- Navigation

Q35. Difference between forward() and include()

forward()	include()
Transfers request	Includes output
Current servlet stops	Current servlet continues
URL unchanged	URL unchanged
One response	Combined response

Q36. What is sendRedirect()?

Answer

sendRedirect()

redirects browser to another URL.

Example

```
response.sendRedirect("welcome.jsp");
```

Browser sends

NEW REQUEST

Q37. Difference between forward() and sendRedirect()

forward()	sendRedirect()
Server side	Client side
Same request	New request
Faster	Slightly slower
URL unchanged	URL changes
Can share request object	Cannot share request object

Q38. Which is faster?

Answer

forward()

is faster because

only one request

is processed.

sendRedirect()

creates

second request.

Q39. When should we use sendRedirect()?

Answer

Use

sendRedirect()

when

- Redirecting another website
- Redirect after login
- Redirect after logout
- Redirect another application

Example

```
response.sendRedirect("https://google.com");
```

Q40. Can forward() redirect to Google?

Answer

No.

forward()

works only

inside

same web application.

For external websites

use

```
sendRedirect()
```

Q41. What is a Filter?

Answer

A **Filter** is a Java component that intercepts client requests and server responses **before they reach a Servlet or JSP** and **after they leave the Servlet or JSP**.

It is used to perform **pre-processing** and **post-processing**.

A Filter acts as an intermediary between the client and the server.

Applications

- Authentication
- Authorization
- Logging
- Data Compression
- Encryption
- Input Validation
- Performance Monitoring

Q42. Why do we use Filters?

Answer

Instead of writing the same code in every Servlet, a Filter allows common functionality to be written once and applied to multiple resources.

Examples include:

- Checking whether a user is logged in.
- Logging request details.
- Compressing responses.
- Blocking unauthorized users.

This improves **code reusability** and **maintainability**.

Q43. What are the advantages of Filters?

Answer

- Reusable code
- Centralized request processing
- Improves security
- Easy logging and auditing
- Reduces duplicate code
- Enhances application performance

Q44. What is the life cycle of a Filter?

Answer

A Filter has **three lifecycle methods**:

1. **init(FilterConfig config)** – Called once when the filter is created.
2. **doFilter(ServletRequest request, ServletResponse response, FilterChain chain)** – Called for every request.
3. **destroy()** – Called once before the filter is removed.

Q45. Explain the `init()` method of a Filter.

Answer

Syntax

```
public void init(FilterConfig config)
```

It is executed **only once**.

It is used to:

- Initialize resources.
- Read configuration parameters.
- Create database connections (if needed).

Q46. Explain the `doFilter()` method.

Answer

The `doFilter()` method is the heart of a Filter.

Syntax

```
public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain chain)
```

It is called for **every client request**.

Typical steps:

1. Receive request.
2. Perform validation or logging.
3. Pass control to the next Filter or Servlet.
4. Modify the response if required.

Q47. Explain the `destroy()` method.

Answer

`destroy()` is called once before the Filter is removed from memory.

It is used to:

- Release resources.
- Close database connections.
- Close files.
- Stop background threads.

Q48. What is FilterChain?

Answer

`FilterChain` is an interface used to invoke the next Filter or the target Servlet.

Method

```
chain.doFilter(request, response);
```

Without calling `chain.doFilter()`, the request will not proceed to the next Filter or Servlet.

Q49. What happens if `chain.doFilter()` is not called?

Answer

The request stops at the current Filter.

The Servlet or JSP is **never executed**.

This technique is commonly used to block unauthorized users.

Example:

- If the user is not logged in, the Filter can redirect them to the login page instead of calling `chain.doFilter()`.

Q50. What is the difference between a Filter and a Servlet?

Filter	Servlet
Intercepts requests/responses	Processes requests
Used before/after Servlets	Generates responses
Cannot directly replace business logic	Contains business logic
Common functionality	Specific functionality

Q51. Can multiple Filters be applied to a single Servlet?

Answer

Yes.

Multiple Filters can be configured for the same Servlet.

The request passes through them in the configured order.

Example flow:

```
Browser
   ↓
Authentication Filter
   ↓
Logging Filter
   ↓
Compression Filter
   ↓
Servlet
```

The response then travels back through the Filters in reverse order.

Q52. What is Filter Mapping?

Answer

Filter Mapping specifies **which URL patterns or Servlets** a Filter should intercept.

Example (`web.xml`):

```
<filter-mapping>
    <filter-name>LoginFilter</filter-name>
    <url-pattern>/admin/*</url-pattern>
</filter-mapping>
```

This Filter runs for all URLs under `/admin/`.

Q53. What is the difference between a Filter and an Interceptor?

Answer

Filter	Interceptor
Part of Servlet API	Part of Spring MVC
Works with Servlet requests	Works with Spring controllers
Configured in `web.xml` or `@WebFilter`	Configured in Spring configuration
Framework-independent	Spring-specific

Q54. Can a Filter modify the request and response?

Answer

Yes.

A Filter can:

- Read request parameters.
- Validate input.
- Add or modify headers.
- Compress output.
- Encrypt/decrypt data.
- Log request and response information.

This makes Filters very useful for cross-cutting concerns.

Q55. Give real-time applications of Filters.

Answer

Common real-world uses include:

- User authentication
- Authorization checks
- Request logging
- Response compression (GZIP)

- Input validation
- Character encoding (UTF-8)
- Measuring execution time
- Preventing SQL Injection and XSS attacks
- CORS handling
- Rate limiting

Q56. What is Session Tracking?

Answer

Session Tracking is a mechanism used to maintain the state of a user across multiple HTTP requests.

Since **HTTP is a Stateless Protocol**, the server cannot remember previous requests automatically.

Session Tracking helps identify the same user during multiple interactions.

Example

```
Login
   ↓
View Profile
   ↓
Update Profile
   ↓
Logout
```

Without session tracking, every page request is treated as a **new user**.

Q57. Why do we need Session Tracking?

Answer

HTTP creates a **new connection** for every request.

Therefore,

- Login information is lost.
- Shopping cart disappears.
- User identity cannot be maintained.

Session Tracking solves these problems.

Q58. What are the different Session Tracking Techniques?

Answer

There are **four major techniques**.

Technique	Server/Client
Cookies	Client
HttpSession	Server
URL Rewriting	Client URL
Hidden Form Fields	HTML Form

Q59. What is a Cookie?

Answer

A **Cookie** is a small text file stored in the user's browser.

It stores

- Username
- Language
- Theme
- Login token
- User preferences

Maximum size is about **4 KB**.

Q60. How do you create a Cookie?

Answer

Example

```
Cookie c = new Cookie("username","Rahul");
response.addCookie(c);
```

The browser stores the cookie.

Q61. How do you read Cookies?

Answer

Example

```
Cookie cookies[] = request.getCookies();

for(Cookie c : cookies)
{
    System.out.println(c.getName());
    System.out.println(c.getValue());
}
```

Q62. Advantages and Disadvantages of Cookies

Advantages

- Simple
- Fast
- Easy to implement
- Stores user preferences

Disadvantages

- Small storage (4 KB)
- Can be deleted by user
- Security issues
- Stored on client machine

Q63. What is HttpSession?

Answer

HttpSession is the most widely used session tracking mechanism.

It stores session data on the **server**.

Example

```
HttpSession session = request.getSession();
```

Q64. Important Methods of HttpSession

Answer

Method	Purpose
getSession()	Create session
setAttribute()	Store object
getAttribute()	Retrieve object
invalidate()	Destroy session
getId()	Session ID
getCreationTime()	Session creation time

Q65. How do you store data in HttpSession?

Answer

Example

```
HttpSession session = request.getSession();

session.setAttribute("username","Rahul");
```

Retrieve

```
String s=(String)session.getAttribute("username");
```

Q66. Difference between Cookies and HttpSession

Cookies	HttpSession
Stored in Browser	Stored in Server
Less Secure	More Secure
4 KB limit	Large storage
User can delete	User cannot directly access
Faster	Slightly slower

Q67. What is URL Rewriting?

Answer

URL Rewriting appends data to the URL.

Example

```
WelcomeServlet?user=Rahul
```

Example

```
response.sendRedirect("Welcome?name=Rahul");
```

Q68. Advantages and Disadvantages of URL Rewriting

Advantages

- Works even when Cookies are disabled.
- Simple.
- No browser storage.

Disadvantages

- URL becomes lengthy.
- Data is visible in the address bar.
- Less secure.

Q69. What are Hidden Form Fields?

Answer

Hidden Form Fields are invisible HTML input fields used to transfer data between pages.

Example

```
<input type="hidden"
       name="username"
       value="Rahul">
```

The browser does not display this field, but it is submitted with the form.

Q70. Compare all Session Tracking Techniques

Feature	Cookies	HttpSession	URL Rewriting	Hidden Fields
Storage	Client	Server	URL	HTML Form
Security	Low	High	Low	Medium
Capacity	4 KB	Large	Small	Small
Visible to User	No	No	Yes	No
Browser Dependency	Yes	No	No	Yes

Q71. What is a Servlet Event?

Answer

A **Servlet Event** is an action or occurrence recognized by the **Servlet Container (Tomcat)** during the execution of a web application.

Whenever an important action occurs (such as application start, session creation, or session destruction), the container generates an event.

Examples:

- Web application starts.
- User logs in (session created).
- User logs out (session destroyed).
- Application stops.

Servlet Events help developers execute code automatically when these events occur.

Q72. What is a Listener?

Answer

A **Listener** is a Java class that listens for Servlet Events and executes predefined code automatically when an event occurs.

A Listener implements one of the Servlet Listener interfaces.

Example uses:

- Logging
- Counting online users

- Initializing resources
- Closing database connections

Q73. Why do we use Listeners?

Answer

Listeners are used to perform automatic tasks without explicitly calling methods.

Applications include:

- Counting active users.
- Loading configuration files.
- Creating database connection pools.
- Logging application start and stop times.
- Cleaning temporary resources.

Q74. What are the types of Servlet Events?

Answer

Servlet Events are mainly of two types:

1. Context-Level Events

Generated for the entire web application.

Examples:

- Application starts.
- Application stops.

Handled by:

- `ServletContextListener`

---

2. Session-Level Events

Generated for each user session.

Examples:

- Session created.
- Session destroyed.

Handled by:

- `HttpSessionListener`

Q75. What is ServletContextListener?

Answer

`ServletContextListener` listens to **application-level events**.

It is invoked when:

- Web application starts.
- Web application shuts down.

Methods:

```
contextInitialized()
```

```
contextDestroyed()
```

Applications:

- Load configuration files.
- Create database connection pools.
- Initialize caches.

Q76. Explain contextInitialized().

Answer

Method:

```
public void contextInitialized(ServletContextEvent event)
```

Called **once** when the web application starts.

Used to:

- Initialize resources.
- Read global configuration.

- Establish shared database connections.
- Load application-wide data.

Q77. Explain contextDestroyed().

Answer

Method:

```
public void contextDestroyed(ServletContextEvent event)
```

Called **once** when the application stops.

Used to:

- Close database connections.
- Release memory.
- Save logs.
- Clean temporary files.

Q78. What is HttpSessionListener?

Answer

`HttpSessionListener` listens to **session-related events**.

Methods:

`sessionCreated()`

`sessionDestroyed()`

Applications:

- Count active users.
- Track login/logout.
- Maintain session statistics.

Q79. Explain sessionCreated().

Answer

Method:

```
public void sessionCreated(HttpSessionEvent event)
```

Called whenever a **new session** is created.

Typical uses:

- Increase online user count.
- Initialize session attributes.
- Log user login time.

Q80. Explain sessionDestroyed().

Answer

Method:

```
public void sessionDestroyed(HttpSessionEvent event)
```

Called when:

- Session timeout occurs.
- User logs out.
- Session is invalidated.

Typical uses:

- Decrease online user count.
- Save user activity.
- Release session resources.

Q81. Difference between Filter and Listener

Filter	Listener
Intercepts requests/responses	Listens to application/session events
Executes for every request	Executes only when an event occurs
Used for authentication, logging	Used for initialization, cleanup, statistics
Uses `doFilter()`	Uses event callback methods

Q82. Difference between ServletContext and ServletContextListener

ServletContext	ServletContextListener
Object representing application context	Interface listening for application events
Stores global data	Responds to application start/stop
Accessed by servlets	Implemented by listener classes

Q83. Can multiple Listeners be used?

Answer

Yes.

A web application can have multiple Listeners.

Examples:

- `ServletContextListener`
- `HttpSessionListener`
- `ServletRequestListener`
- `HttpSessionAttributeListener`

The Servlet Container invokes them as required.

Q84. Real-time applications of Listeners

Answer

Listeners are commonly used for:

- Online user counting.
- Audit logging.
- Performance monitoring.
- Resource initialization.
- Resource cleanup.
- Cache loading.
- Email notifications.
- Startup configuration.

Q85. Which Listener is used for online user counting?

Answer

The **HttpSessionListener** is commonly used.

Logic:

- Increment counter in `sessionCreated()`.
- Decrement counter in `sessionDestroyed()`.

Example:

```
public class OnlineUserListener implements HttpSessionListener {

    private static int count = 0;

    public void sessionCreated(HttpSessionEvent se) {
        count++;
    }

    public void sessionDestroyed(HttpSessionEvent se) {
        count--;
    }

    public static int getCount() {
        return count;
    }
}
```

This is a classic interview question.