Unit-III: Servlet API and Overview

1. Servlet Introduction

A **Servlet** is a server-side Java programming component managed by a Web Container (such as Apache Tomcat). Servlets extend the capabilities of web servers by dynamically generating responses (HTML, JSON, XML) for incoming HTTP requests.

Java EE (javax**) vs. Jakarta EE (**jakarta**) Evolution**

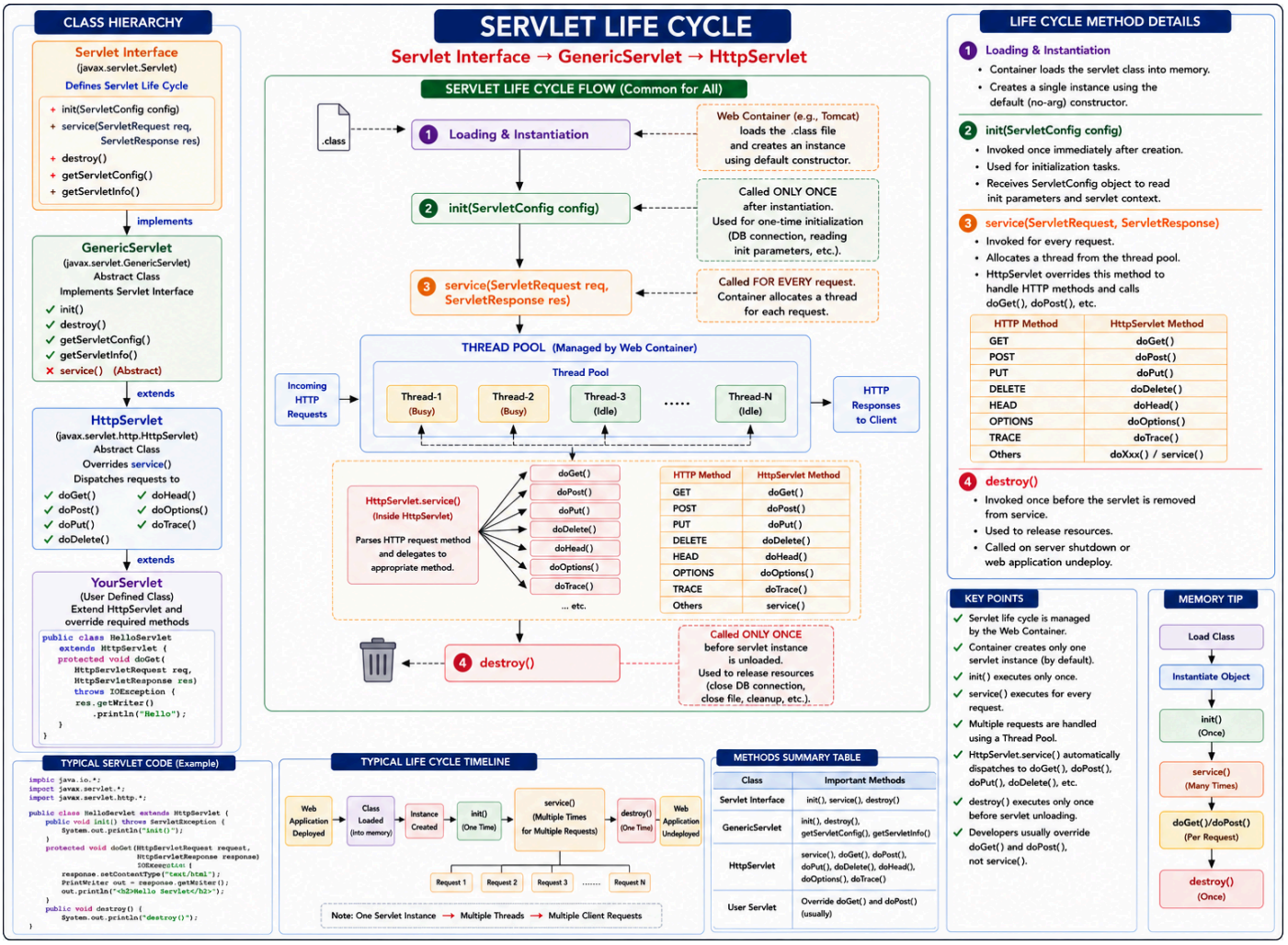
- **Namespace Transition:** Oracle transferred Java EE to the Eclipse Foundation. Modern enterprise Java (Jakarta EE 9+) transitioned package namespaces from javax.servlet.* to jakarta.servlet.*.
- *Note for Exams:* Most university curricula still use javax.servlet.*, but knowing jakarta.servlet.* demonstrates current industry awareness.

Advantages of Servlets over CGI (Common Gateway Interface)

Feature	**CGI (Common Gateway Interface)**	**Java Servlets**
Process Model	Spawns a new OS process per request.	Allocates a lightweight thread per request.
Performance	Slow due to high process creation overhead.	Fast due to thread pooling and memory sharing.
Resource Usage	High memory consumption under heavy load.	Extremely efficient memory utilization.
Portability	Platform-dependent scripts (C/C++, Perl).	"Write Once, Run Anywhere" Java standard.

2. Servlet Life Cycle (SLC)

The Servlet life cycle is entirely controlled by the **Web Container**. It consists of three primary phase methods defined in the javax.servlet.Servlet interface:



Life Cycle Methods

1. **Loading & Instantiation:** The container loads the .class file into memory and creates an instance using its default constructor.
2. init(ServletConfig config)**:**
    - Invoked **once** immediately after instantiation.
    - Used for one-time initialization tasks (e.g., establishing database connections, reading startup parameters).
3. service(ServletRequest req, ServletResponse res)**:**
    - Invoked **for every incoming request**.
    - Spawns or allocates a thread from the container pool.
    - HttpServlet overrides service() to parse HTTP methods and delegate execution to doGet(), doPost(), doPut(), doDelete(), etc.
4. destroy()**:**
    - Invoked **once** when the web container shuts down or unloads the servlet.

- Used to release persistent resources (e.g., closing database connections, flushing file streams).

3. Thread Safety & SingleThreadModel

Are Servlets Thread-Safe?

- **Default Behavior: No.** The Web Container instantiates **only ONE instance** of a Servlet class per web application. Every incoming HTTP request triggers a new thread executing concurrently on that **single shared instance**.
- **Thread-Safety Golden Rule:** Never declare mutable instance variables inside a Servlet class! Store request-specific data strictly inside **local variables** within doGet() or doPost(), which are safely stored on the thread stack.
- **Legacy Note (**SingleThreadModel**):** In early Servlet versions, implementing SingleThreadModel forced the container to ensure no two threads executed service() simultaneously. This interface was **deprecated** because it degraded server performance drastically under high user traffic.

4. Types of Servlets

Java provides two main abstract classes for implementing servlets:

1. GenericServlet (javax.servlet.GenericServlet**)**

- Protocol-independent abstract class implementing Servlet and ServletConfig.
- Requires overriding the abstract service(ServletRequest req, ServletResponse res) method.
- Rarely used for web development because it does not natively parse HTTP methods.

2. HttpServlet (javax.servlet.http.HttpServlet**)**

- Protocol-dependent abstract class extending GenericServlet specifically tailored for HTTP.
- Provides built-in HTTP convenience methods (doGet, doPost, doPut, doDelete).

5. Servlet Configuration: Annotations (Servlet 3.0+) vs. web.xml

Modern Annotation-Based Configuration

Since Servlet 3.0+, developers can use annotations directly on Java classes, eliminating XML boilerplate in web.xml.

```
import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebInitParam;

import javax.servlet.annotation.WebServlet;
```

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(
    urlPatterns = {"/configTest", "/legacyConfig"},
    loadOnStartup = 1,
    initParams = {
        @WebInitParam(name = "adminEmail", value = "admin@example.com")
    }
)
public class ConfigServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        String adminEmail = getServletConfig().getInitParameter("adminEmail");
        resp.getWriter().println("Admin Email: " + adminEmail);
    }
}
```

Legacy Deployment Descriptor (web.xml**) Configuration**

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="4.0">

    <!-- Servlet Declaration -->
    <servlet>
        <servlet-name>ConfigServlet</servlet-name>
        <servlet-class>com.example.ConfigServlet</servlet-class>
        <init-param>
```

```
        <param-name>adminEmail</param-name>
        <param-value>admin@example.com</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<!-- Mapping to URL -->
<servlet-mapping>
    <servlet-name>ConfigServlet</servlet-name>
    <url-pattern>/configTest</url-pattern>
</servlet-mapping>

<!-- Global Application Parameter -->
<context-param>
    <param-name>appName</param-name>
    <param-value>Enterprise Web Portal</param-value>
</context-param>

</web-app>
```

6. Working with ServletContext **and** ServletConfig

Feature	**ServletConfig**	**ServletContext**
Scope	One instance per **Servlet**.	One instance per **Web Application**.
Accessibility	Accessible only within the specific configured servlet.	Shared across all servlets/filters in the entire web application.
Configuration	Declared inside @WebInitParam or <init-param>.	Declared inside <context-param>.

Feature	ServletConfig	ServletContext
Use Case	Servlet-specific configs (e.g., local file path).	Global configs (e.g., app-wide database pool parameters).

7. Servlet Attributes & Scope Management

Attributes are key-value Java Objects stored inside container scopes to share state during execution.

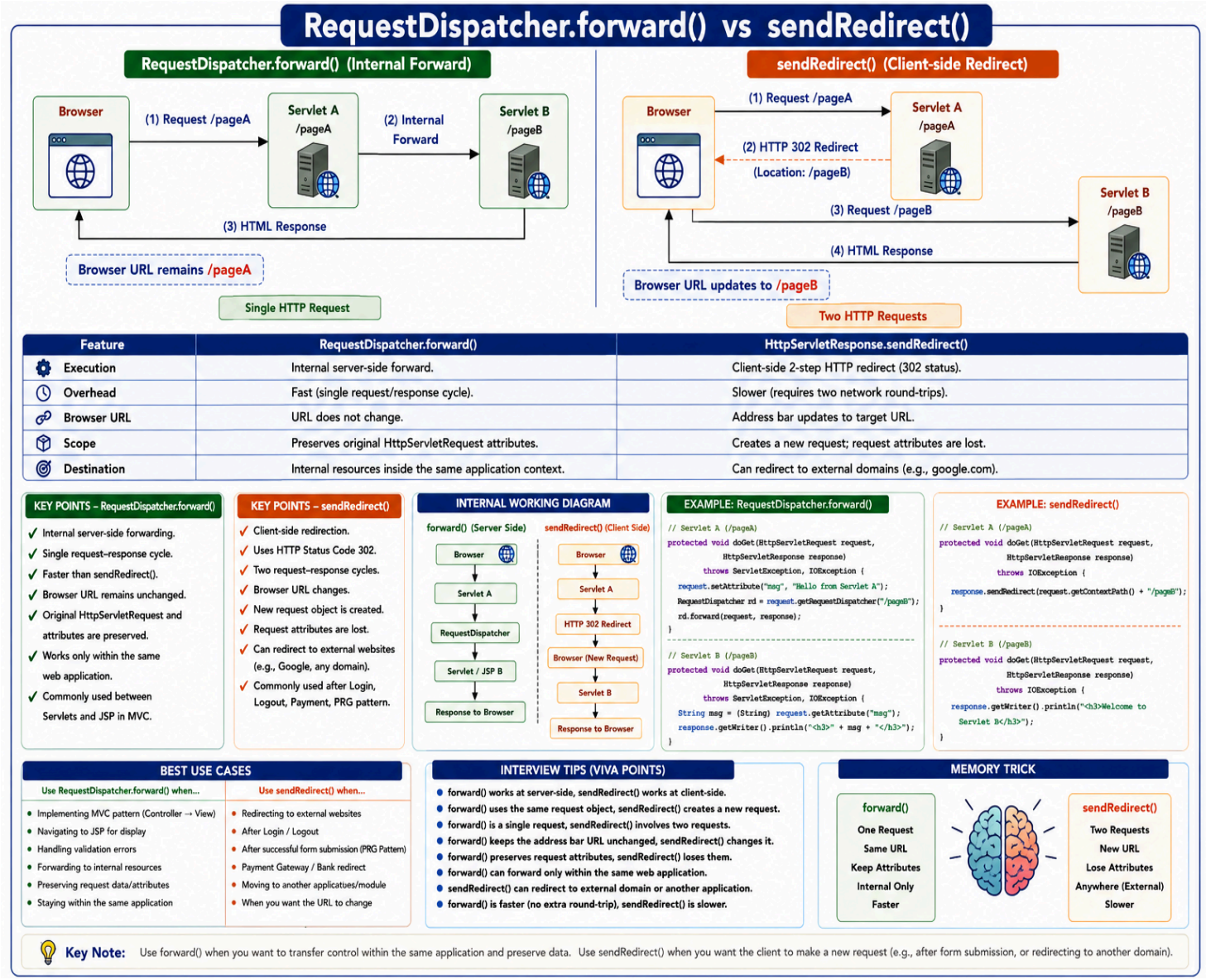
Scope Hierarchy

Scope Level	Interface	Lifetime	Typical Use Case
Request Scope	HttpServletRequest	Single request/response lifecycle.	Passing data during RequestDispatcher.forward().
Session Scope	HttpSession	Spans multiple requests from a specific user browser session.	Shopping cart items, logged-in user profile.
Application Scope	ServletContext	Spans application startup to container shutdown.	Global counter, shared application config.

Shared Manipulation Methods

- setAttribute(String name, Object value) — Store object.
- getAttribute(String name) — Retrieve object (requires explicit typecasting).
- removeAttribute(String name) — Remove object.

8. RequestDispatcher **vs** sendRedirect()



Feature	RequestDispatcher.forward()	HttpServletResponse.sendRedirect()
Execution	Internal server-side forward.	Client-side 2-step HTTP redirect (302 status).
Overhead	Fast (single request/response cycle).	Slower (requires two network round-trips).
Browser URL	URL does **not** change.	Address bar updates to target URL.

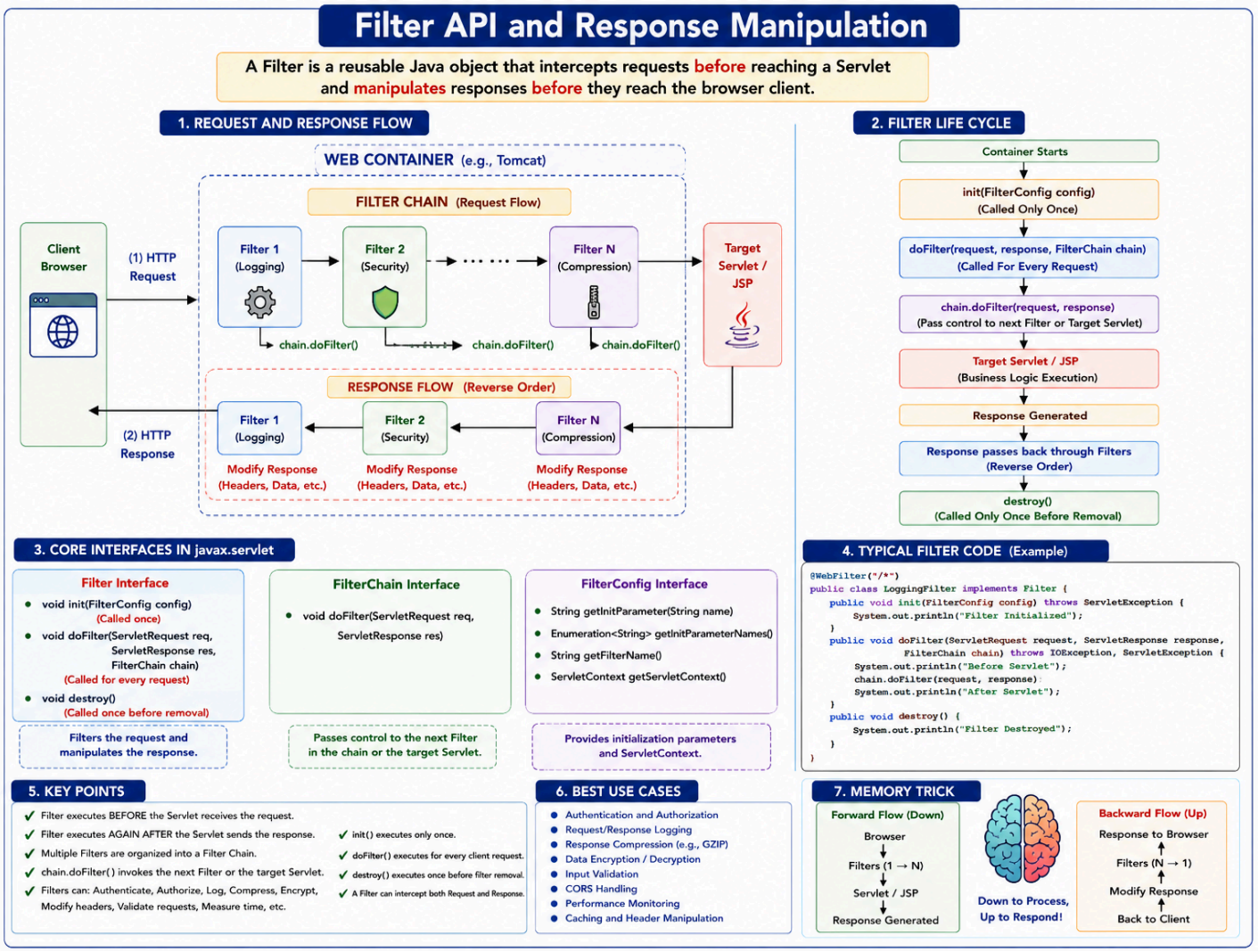
Feature	RequestDispatcher.forward()	HttpServletResponse.sendRedirect()
Scope	Preserves original HttpServletRequest attributes.	Creates a **new** request; request attributes are lost.
Destination	Internal resources inside the same application context.	Can redirect to external domains (e.g., google.com).

9. Filter API and Response Manipulation

A **Filter** is a reusable Java object that intercepts requests before reaching a Servlet and manipulates responses before they reach the browser client.

Core Interfaces in javax.servlet

1. Filter: Must implement init(), doFilter(), and destroy().
2. FilterChain: Calls the next filter via chain.doFilter(request, response).
3. FilterConfig: Used to read initialization parameters.



Code Example: Response Header & Security Filter

```
package com.example;
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletResponse;

@WebFilter("/*") // Intercepts all incoming requests
public class SecurityFilter implements Filter {
   @Override
   public void init(FilterConfig filterConfig) throws ServletException {}
   @Override
   public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
         throws IOException, ServletException {

      HttpServletResponse httpResponse = (HttpServletResponse) response;

      // Manipulate response headers before servlet processing
      httpResponse.setHeader("X-Frame-Options", "DENY");
      httpResponse.setHeader("X-Content-Type-Options", "nosniff");

      // Pass request down the chain
      chain.doFilter(request, response);
```

```
    }

    @Override
    public void destroy() {}
}
```

10. Session Tracking Mechanisms & Security

HTTP is stateless. Session tracking techniques associate consecutive client requests with an identity state.

1. Cookies & Security Flags

Java

```
Cookie userCookie = new Cookie("userRole", "admin");
userCookie.setMaxAge(60 * 60); // 1 hour expiration

// SECURITY BEST PRACTICES FOR COOKIES
userCookie.setHttpOnly(true); // Prevents client-side JS XSS access
userCookie.setSecure(true);   // Forces HTTPS transmission only

response.addCookie(userCookie);
```

2. HttpSession **API**

Java

```
// Obtain or create session
HttpSession session = request.getSession(true);
session.setAttribute("userProfile", userObj);

// Invalidate session upon logout
session.invalidate();
```

3. Hidden Form Fields

HTML

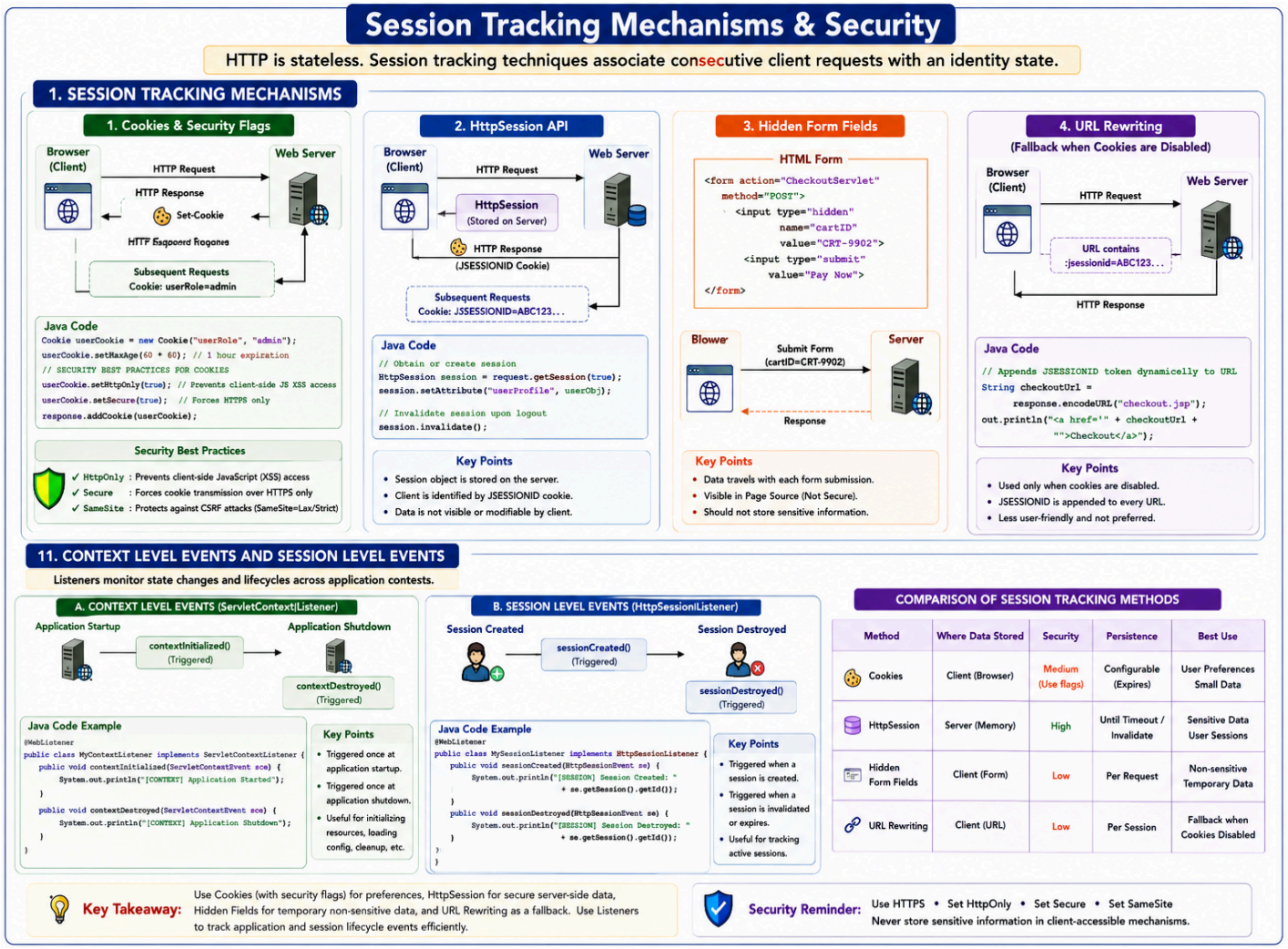
```
<form action="CheckoutServlet" method="POST">
```

```
    <input type="hidden" name="cartID" value="CRT-9902">
    <input type="submit" value="Pay Now">
</form>
```

4. URL Rewriting (Fallback when Cookies are Disabled)

Java

```
// Appends JSESSIONID token dynamically to URL
String checkoutUrl = response.encodeURL("checkout.jsp");
out.println("<a href='" + checkoutUrl + "'>Checkout</a>");
```



11. Context Level Events and Session Level Events

Listeners monitor state changes and lifecycles across application contexts.

Primary Listener Types

- @WebListener ServletContextListener: Triggers on application startup (contextInitialized) and shutdown (contextDestroyed).
- @WebListener HttpSessionListener: Triggers on session creation (sessionCreated) and expiration (sessionDestroyed).

Code Example: Global Monitoring Listener

```
package com.example;

import javax.servlet.ServletContextEvent;

import javax.servlet.ServletContextListener;

import javax.servlet.annotation.WebListener;

import javax.servlet.http.HttpSessionEvent;

import javax.servlet.http.HttpSessionListener;


@WebListener

public class AppEventListener implements ServletContextListener, HttpSessionListener {


   private static int activeUserCount = 0;


   @Override

   public void contextInitialized(ServletContextEvent sce) {

      System.out.println("System Startup: Web Application initialized.");

   }


   @Override

   public void contextDestroyed(ServletContextEvent sce) {

      System.out.println("System Shutdown: Web Application stopped.");

   }


   @Override

   public void sessionCreated(HttpSessionEvent se) {

      activeUserCount++;
```

```
        System.out.println("User Logged In. Current Active Users: " + activeUserCount);
    }

    @Override
    public void sessionDestroyed(HttpSessionEvent se) {
        if (activeUserCount > 0) activeUserCount--;
        System.out.println("User Logged Out. Current Active Users: " + activeUserCount);
    }
}
```