Java Servlets:

Java Servlets are server-side components used to extend the capabilities of a web server. They are commonly used in web applications to process HTTP requests and generate dynamic web content. Here's an explanation of Java Servlets in a point-wise manner with an example for each point:

1. **Servlet Basics:**
   - A Java Servlet is a Java program that runs on a web server and processes client requests.
   - It extends the `javax.servlet.HttpServlet` class and overrides its methods to handle HTTP requests and responses.

   Example:
   ```java
   import javax.servlet.*;
   import javax.servlet.http.*;
   import java.io.IOException;

   public class HelloServlet extends HttpServlet {
       public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
           response.setContentType("text/html");
           PrintWriter out = response.getWriter();
           out.println("<html><body>");
           out.println("<h1>Hello, Servlet!</h1>");
           out.println("</body></html>");
       }
   }
   ```

2. **Deployment and Configuration:**
   - Servlets are packaged in a `.war` file and deployed to a web server or servlet container like Apache Tomcat.
   - Configuration is done using the `web.xml` file or annotations (in modern servlet development).

3. **Request Handling:**
   - Servlets can handle various HTTP methods like GET, POST, PUT, DELETE, etc., depending on the application's requirements.
   - Request parameters and data are accessible through the `HttpServletRequest` object.

   Example:
   ```java
   String username = request.getParameter("username");
   ```

```

4. **Response Generation:**
   - Servlets generate dynamic content by writing to the `HttpServletResponse` object's output stream.
   - You can set response headers, content type, and write HTML or other data to the response.

5. **Session Management:**
   - Servlets can manage user sessions to maintain state between multiple requests using the `HttpSession` object.

   **Example:**
   ```java
   HttpSession session = request.getSession();
   session.setAttribute("user", "John");
   ```

6. **Servlet Lifecycle:**
   - Servlets have a lifecycle with methods like `init()`, `service()`, and `destroy()` that are called at specific times during their execution.

   Example:
   ```java
   public void init() throws ServletException {
       // Initialization code here
   }
   ```

7. **URL Mapping:**
   - Servlets are mapped to URL patterns in the `web.xml` or via annotations to specify which URLs they should handle.

   Example (web.xml):
   ```xml
   <servlet-mapping>
       <servlet-name>HelloServlet</servlet-name>
       <url-pattern>/hello</url-pattern>
   </servlet-mapping>
   ```

8. **Filtering:**
   - Servlet filters can be used to intercept and process requests and responses globally, performing tasks like authentication, logging, or data transformation.
   ```

Example:
```java
public class LoggingFilter implements Filter {
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
        // Logging code here
        chain.doFilter(request, response);
    }
}
```

9. **Exception Handling:**
   - You can handle exceptions in servlets and specify error pages in the `web.xml` for different error codes.

Example (web.xml):
```xml
<error-page>
    <error-code>404</error-code>
    <location>/error404.jsp</location>
</error-page>
```

10. **Security:**
    - Servlets can be secured using various authentication and authorization mechanisms, and access control can be defined in the `web.xml` or through annotations.

Example (web.xml):
```xml
<security-constraint>
    <!-- Security constraints configuration -->
</security-constraint>
```

Java Servlets are a fundamental part of Java-based web applications, providing a powerful means to handle web requests and generate dynamic content.