**Java Servlets** are programs that run on an Web Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

Servlets can be created using the javax.servlet and javax.servlet.http packages , which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

**What does the Servlets do:**

Read the explicit data sent by the clients (browsers)

Read the implicit HTTP request data sent by the clients (browsers)

Process the data and generate the results

Send the explicit data (i.e., the document) to the clients (browsers)

Send the implicit HTTP response to the clients (browsers).

Servlets

* Java Object based on the Servlet API
* Runs in a server application to answer client requests; technically, servlets are not tied to a specific client-server protocol, but they are most commonly used with HTTP and the term „servlet‟ is often used in the context of an “HTTP Servlet”
* Web-tier components in the Java EE architecture.
* Runs in, and is managed by, a web-tier container called the „Servlet Container‟ 
* Mapped to URLs to which clients send requests
* Typically asked with (among other things)
* Processing and/or storing data submitted vial HTML forms
* Generating dynamic content
* javax.servlet
  + Servlet, GenericServlet
  + ServletRequest, ServletResponse
  + ServletConfig, ServletContext
  + RequestDispatcher
* javax.servlet.http
* HttpServlet
* HttpServletRequest
* HttpServletResponse
* HttpSession
* Cookie
* Servlet Processing
* Client sends a request to a web server URL that is mapped to a servlet. Web server passes on the request to the servlet container
  + Servlet container checks if servlet is already loaded
  + If it is not yet loaded, servlet container loads the servlet class and instantiates the servlet, and calls its init method.
  + Servlet container invokes the servlet‟s service method, passing request and response objects as arguments
  + Servlet processes the request using the response object to create the response, which is returned by the servlet container to the web server, which in turn sends the response to the client
  + Subsequent request to the servlet will not require servlet re-instantiation, unless the servlet has been unloaded; before a servlet is unloaded, the servlet container invokes its destroy method
* init(config)
* Invoked once on the servlet by the servlet container when the servlet is instantiated; can be used by the servlet for one-time startup initialization
* service(request, response)
* Invoked each time the servlet is called upon to process a request (typically on a separate thread for each call)
* In HttpServlet, the default Service implementation maps the call to a specific doXXX() method (e.g. doGet, doPost) which is typically overridden to affect the servlet‟s functionality
* Destroy()
* Invoked on the servlet by the servlet container when the servlet is to be unloaded (e.g. when the application is stopped or undeployed); can be used by the servlet for clean-up processing (e.g. resource deallocation)
* Servlet Request Processing (HttpServletRequest)
* Retrieving user-supplied request parameters
* Retrieving request header values
* Servlet Response Processing (HttpServletResponse)
* Setting response status code
* Setting response headers
* Obtaining output object for sending the response
* Servlet Request Dispatching (RequestDispatcher)
* Obtain a RequestDispatcher to a resource (static or dynamic) from the request object

RequestDispatcher rqstDsp;

rqstDsp = request.getRequestDispatcher(res);

* Include the dispatcher resource (or its output) in the current response; one or more resources can be included (e.g. use for banners, footers, etc.)

rqstDsp.include(request, response);

* Forwards the processing of the current request to the dispatcher resource; the servlet processing the current request must not generate a response ( e.g. use in MVC “controller” servlets)

rqstDsp.forward(request, response);

* Session Tracking(HttpSession)
* Session tracking support is implemented either cookies or URL-rewriting o Obtaining session object from the current request

HttpSession session;

session = request.getSession(createNew);

* Obtaining session information (HttpSession)
* getCreationTime(), getLastAccessedTime(), getMaxInactiveInternal(), getId(), isNew(), setMaxInactiveInterval(int val)
* destroying sessions
* invalidate()
* URL-rewriting(HttpServletResponse)
* encodeURL(String url), encodeRedirectURL(String url)