ServletContext

When the servletcontainer (like [Apache Tomcat](http://tomcat.apache.org/)) starts up, it will deploy and load all webapplications. When a webapplication get loaded, the servletcontainer will create the [ServletContext](http://docs.oracle.com/javaee/7/api/javax/servlet/ServletContext.html) once and keep in server's memory. The webapp's web.xml will be parsed and every <servlet>, <filter>and <listener> found in web.xml, or annotated with respectively @WebServlet, @WebFilter and @WebListener, will be created once and kept in server's memory as well. For all filters, the init()method will also be invoked immediately. When the servletcontainer shuts down, it will unload all webapplications, invoke the destroy() of all initialized servlets and filters, and finally the ServletContext and all Servlet, Filter and Listener instances will be trashed.

When the Servlet in question has a <servlet><load-on-startup> or @WebServlet(loadOnStartup)value greater than 0, then its init() method will also immediately be invoked during startup. Those servlets are initialized in the same order as "load-on-startup" value represents, or if they are the same, then the order in the web.xml or @WebServlet classloading. Or, if the "load-on-startup" value is absent, then the init() method will only be invoked on very first HTTP request hitting the servlet in question.

HttpServletRequest and HttpServletResponse

The servletcontainer is attached to a webserver which listens on HTTP requests on a certain port number, which is usually 8080 in development and 80 in production. When a client (user with a webbrowser) sends a HTTP request, the servletcontainer will create new [HttpServletRequest](http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletRequest.html) and [HttpServletResponse](http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletResponse.html) objects and pass it through the methods of the already-created Filter and Servlet instances whose url-pattern matches the request URL, all in the same thread.

The request object provides access to all information of the HTTP request, such as the request headers and the request body. The response object provides facility to control and send the HTTP response the way you want, such as setting headers and the body (usually with HTML content from a JSP file). When the HTTP response is committed and finished, then both the request and response objects will be trashed.

HttpSession

When a client visits the webapp for the first time and/or the [HttpSession](http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpSession.html) is to be obtained for the first time by request.getSession(), then the servletcontainer will create it, generate a long and unique ID (which you can get by session.getId()) and store it in server's memory. The servletcontainer will also set a [Cookie](http://docs.oracle.com/javaee/7/api/javax/servlet/http/Cookie.html) in the Set-Cookie header of the HTTP response with JSESSIONID as cookie name and the unique session ID as cookie value.

As per the [HTTP cookie specification](http://www.faqs.org/rfcs/rfc2965.html) (a contract a decent webbrowser and webserver has to adhere), the client (the webbrowser) is required to send this cookie back in the subsequent requests in the Cookie header as long as the cookie is valid. Using browser builtin HTTP traffic monitor you can check them (press F12 in Chrome / Firefox23+ / IE9+ and check *Net/Network* tab). The servletcontainer will determine the Cookie header of every incoming HTTP request for the presence of the cookie with the name JSESSIONID and use its value (the session ID) to get the associated HttpSession from server's memory.

The HttpSession lives until it has not been used for more than the <session-timeout> time, a setting you can specify in web.xml, which defaults to 30 minutes. So when the client doesn't visit the webapp anymore for over 30 minutes, then the servletcontainer will trash the session. Every subsequent request, even though with the cookie specified, will not have access to the same session anymore. The servletcontainer will create a new one.

On the other hand, the session cookie on the client side has a default lifetime which is as long as the browser instance is running. So when the client closes the browser instance (all tabs/windows), then the session will be trashed at the client side. In a new browser instance the cookie associated with the session won't be sent anymore. A new request.getSession() would return a brand new HttpSession and set a cookie with a brand new session ID.

In a nutshell

* The ServletContext lives as long as the webapp lives. It's been shared among *all* requests in *all*sessions.
* The HttpSession lives as long as the client is interacting with the webapp with the same browser instance and the session hasn't timed out at the server side yet. It's been shared among*all* requests in the *same* session.
* The HttpServletRequest and HttpServletResponse lives as long as the client has sent it until the complete response (the webpage) is arrived. It is *not* being shared elsewhere.
* Any Servlet, Filter and Listener lives as long as the webapp lives. They are being shared among *all* requests in *all* sessions.
* Any attribute which you set in ServletContext, HttpServletRequest and HttpSession will live as long as the object in question lives.

Threadsafety

That said, your major concern is possibly *threadsafety*. You should now have learnt that Servlets and filters are shared among all requests. That's the nice thing of Java, it's multithreaded and different threads (read: HTTP requests) can make use of the same instance. It would otherwise have been too expensive to recreate it on every request.

But you should also realize that you should **never** assign any request or session scoped data as an*instance* variable of a servlet or filter. It will be shared among all other requests in other sessions. That's *threadunsafe*! The below example illustrates that:

public class ExampleServlet extends HttpServlet {

private Object thisIsNOTThreadSafe;

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

Object thisIsThreadSafe;

thisIsNOTThreadSafe = request.getParameter("foo"); // BAD!! Shared among all requests!

thisIsThreadSafe = request.getParameter("foo"); // OK, this is thread safe.

}

}