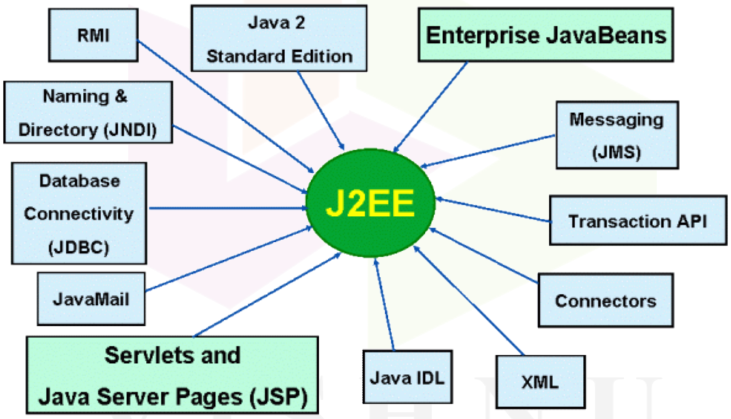
Servlets

Java Enterprise Edition (JEE)

- JEE provides a platform for developing enterprise applications which are portable and scalable.
- JEE application server provides functionality for transaction management, security, scalability, concurrency and others.
- JEE provides API specifications like: JDBC, RMI, JMS, e-mail, web services, XML, Enterprise JavaBeans, Connectors, Servlets, Java Server Pages (JSP) and others.

J2EE Technologies



J2EE

RMI

Java 2 Standard Edition

Enterprise JavaBeans

Naming & Directory (JNDI)

Messaging (JMS)

Database Connectivity (JDBC)

Transaction API

JavaMail

Connectors

Servlets and Java Server Pages (JSP)

Java IDL

XML

Tools or Applications Needed

- JDK 1.6 or later
- Apache Tomcat 7.0.xx
- Text Editor

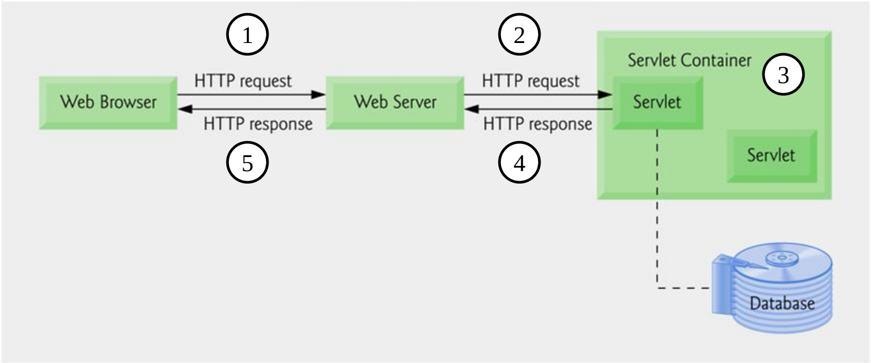
  or

- Eclipse with Apache Tomcat

Introduction

- A servlet is a special java class which is dynamically loaded on the server and used to generate dynamic content.
- Process:
  - Client sends a request to the web server.
  - Web server searches and initiates the servlet.
  - Servlet processes the request and forwards the response to the web server.

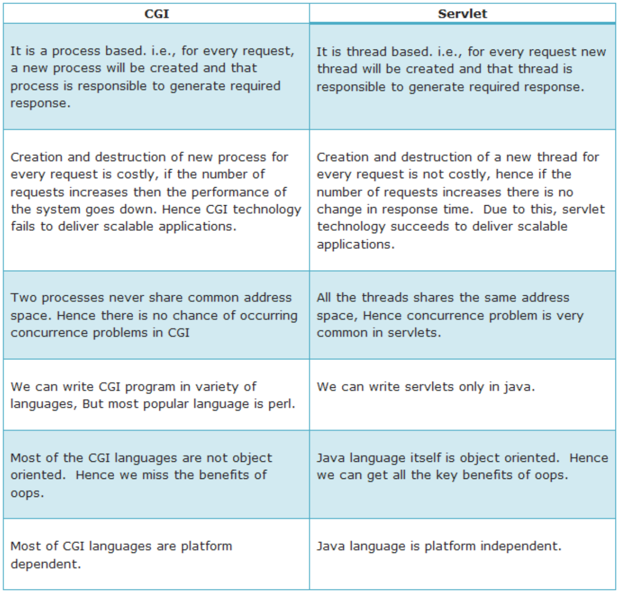
Servlet Architecture



Features of Servlet Technology

- High level
- Component-based
- Platform independent
- Server independent
- High performance (overcomes CGI limitation)
- Highly scalable (overcomes CGI limitation)

CGI Vs Servlets



CGI	Servlet
It is a process based. i.e., for every request, a new process will be created and that process is responsible to generate required response.	It is thread based. i.e., for every request new thread will be created and that thread is responsible to generate required response.
Creation and destruction of new process for every request is costly, if the number of requests increases then the performance of the system goes down. Hence CGI technology fails to deliver scalable applications.	Creation and destruction of a new thread for every request is not costly, hence if the number of requests increases there is no change in response time. Due to this, servlet technology succeeds to deliver scalable applications.
Two processes never share common address space. Hence there is no chance of occurring concurrence problems in CGI	All the threads shares the same address space, Hence concurrence problem is very common in servlets.
We can write CGI program in variety of languages, But most popular language is perl.	We can write servlets only in java.
Most of the CGI languages are not object oriented. Hence we miss the benefits of oops.	Java language itself is object oriented. Hence we can get all the key benefits of oops.
Most of CGI languages are platform dependent.	Java language is platform independent.

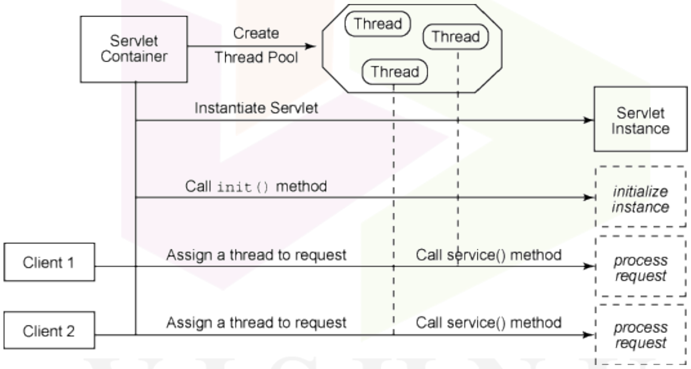
Servlet Container (Servlet Engine)

- Servlet container or servlet engine is a set of objects that provides the runtime environment for servlets.
- Servlet container manages the life cycle of servlet.

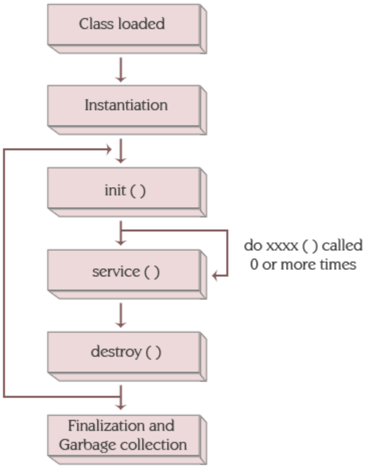
Servlet Container (cont…)

- Servlet container provides the following functionalities:
  - Network services
  - Decode and encode MIME based messages
  - Manage servlet life cycle
  - Resource management
  - Security service
  - Session management

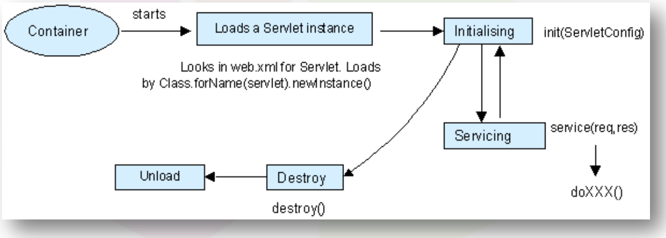
Servlet Life Cycle



Servlet Life Cycle (cont…)



Servlet Life Cycle (cont…)



Servlet Life Cycle (cont…)

- Stages of the servlet life cycle
  - Loading a servlet
  - Initializing a servlet
  - Request handling
  - Destroying a servlet

Loading a Servlet

- Involves loading of servlet by the servlet container.
- Servlet container performs *loading (local file system or remote)* and *instantiation* as part of this stage.

Initializing a Servlet

- After the instantiation of the servlet, the servlet container initializes the instantiated servlet object.
- Servlet container invokes the init(ServletConfig) method on the servlet object.
- May raise *ServletException* or *UnavailableException.*

Request Handling

- After loading and initializing the servlet, the servlet container creates the objects for *ServletRequest* and *ServletResponse*.
- If the request is a HTTP request, servlet container creates the objects for *HTTPServletRequest* and *HTTPServletResponse*.
- After creating the objects, container invokes service() method to serve the request of the client.
- At this stage *ServletException* or *UnavailableException* or *IOException* might be raised.

Destroying a Servlet

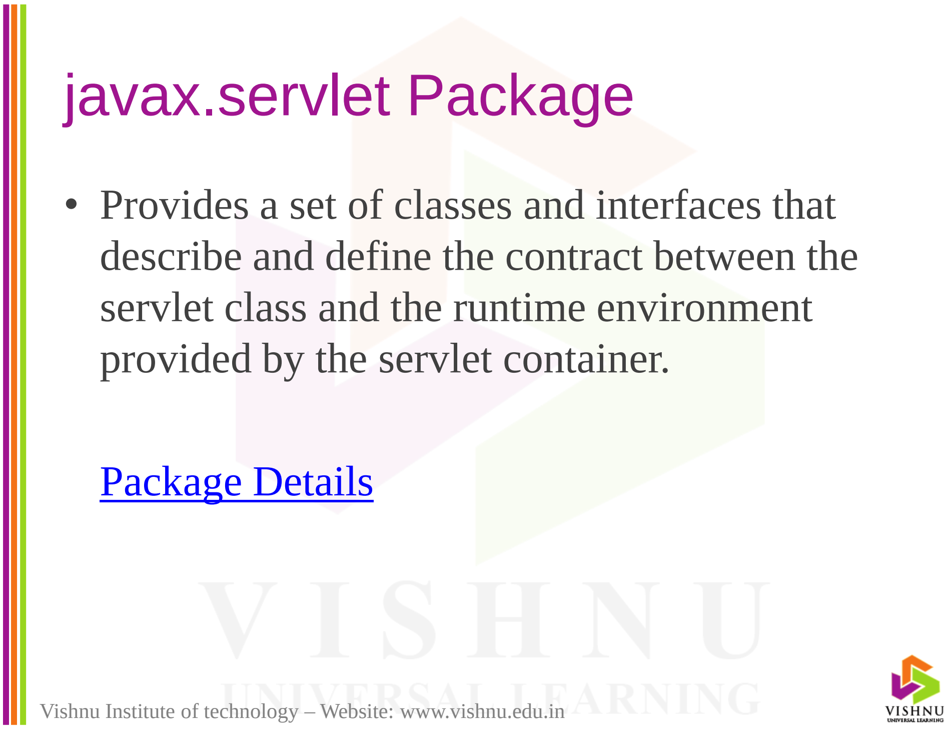
- The following are the conditions under which the container decides to destroy the servlet:
  - When the context is unloaded (shutdown).
  - When the service() method throws *UnavailableException.*
- When the container decides to destroy the servlet, it allows all the current running threads to complete and stops any further requests.
- Finally destroy() method is invoked.

Implementing a Servlet Object

- A servlet is a normal java class which follows these set of rules:
  - Should be a public non-abstract class.
  - Should be a subtype of *javax.servlet.Servlet* interface or *HttpServlet* class.
  - Should contain a zero or no argument constructor.
  - The inherited methods from the *Servlet* interface should not be declared as *final*.

Servlet API

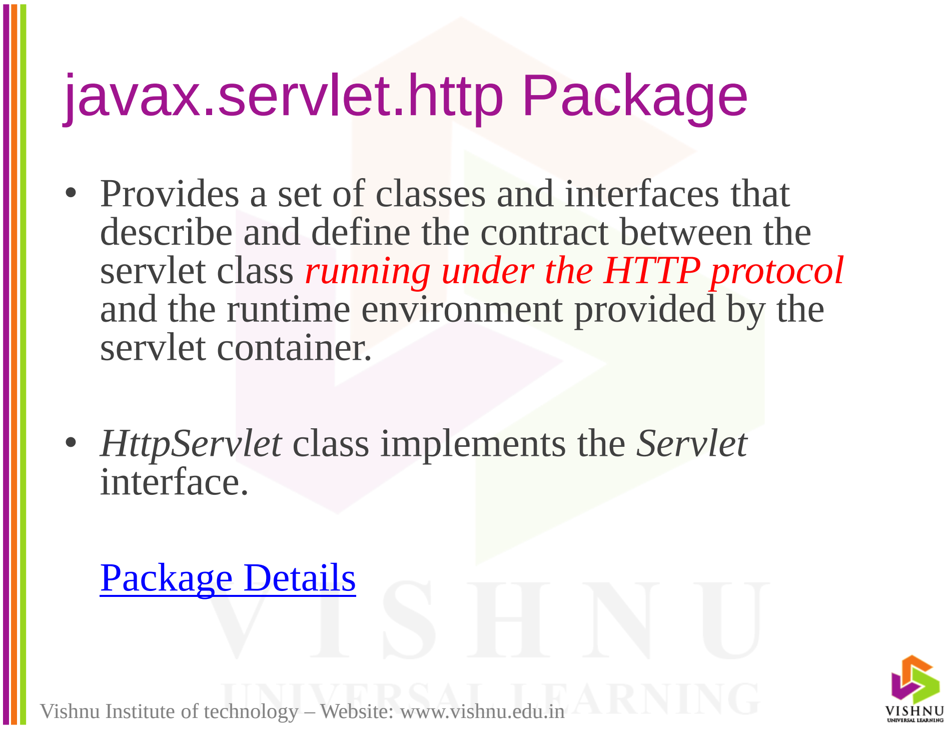
- Servlet API is a set of classes and interfaces that specify a contract between a servlet class and a servlet container.
- There are many servlet containers like Tomcat, Weblogic, JBoss etc.
- These classes and interfaces are grouped into two packages:
  - javax.servlet
  - javax.servlet.http



javax.servlet Package

- Provides a set of classes and interfaces that describe and define the contract between the servlet class and the runtime environment provided by the servlet container.

[Package Details]



javax.servlet.http Package

- Provides a set of classes and interfaces that describe and define the contract between the servlet class *running under the HTTP protocol* and the runtime environment provided by the servlet container.
- *HttpServlet* class implements the *Servlet* interface.

[Package Details]

Creating and Executing a Servlet

- Basic steps:
  1. Create a HTML page
  2. Create the Servlet file
  3. Create the web.xml (deployment descriptor) file
  4. Deploy the application

Creating a HTML Page

index.html

```
<html>
   <head>
        <title>Hello Servlet</title>
   </head>
   <body>
        <form  action="HelloServlet" method="get">
                <input type="submit" value="Invoke Servlet!" />
        </form>
   </body>
</html>
```

Creating Servlet File

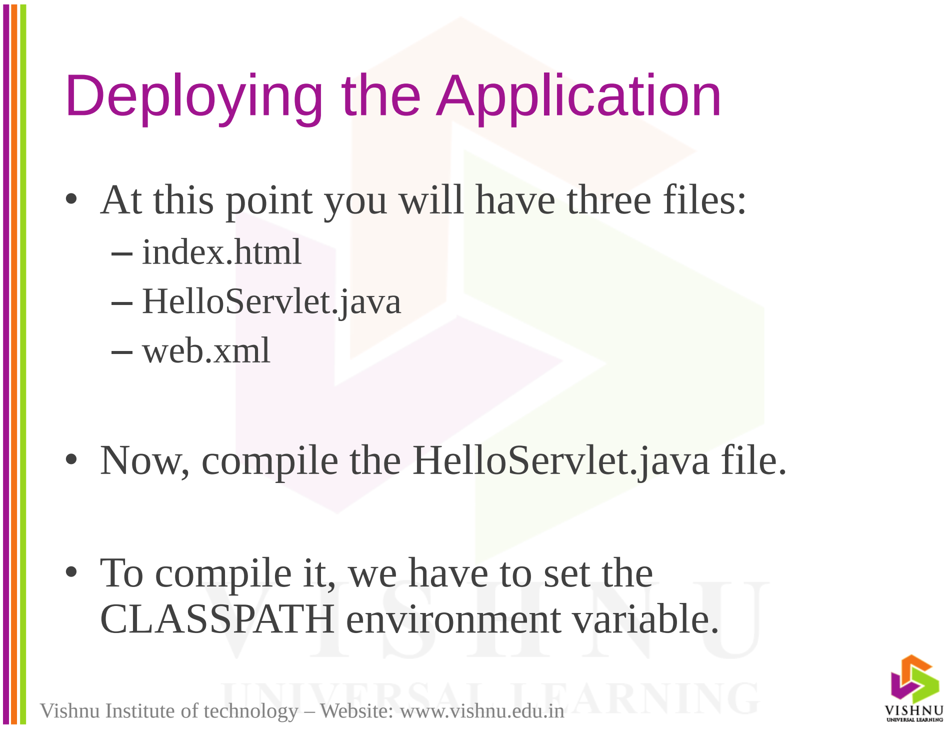
HelloServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloServlet extends HttpServlet
{
            public void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException
            {
                        response.getWriter().write("Hello World!");
            }
}
```

Creating web.xml File

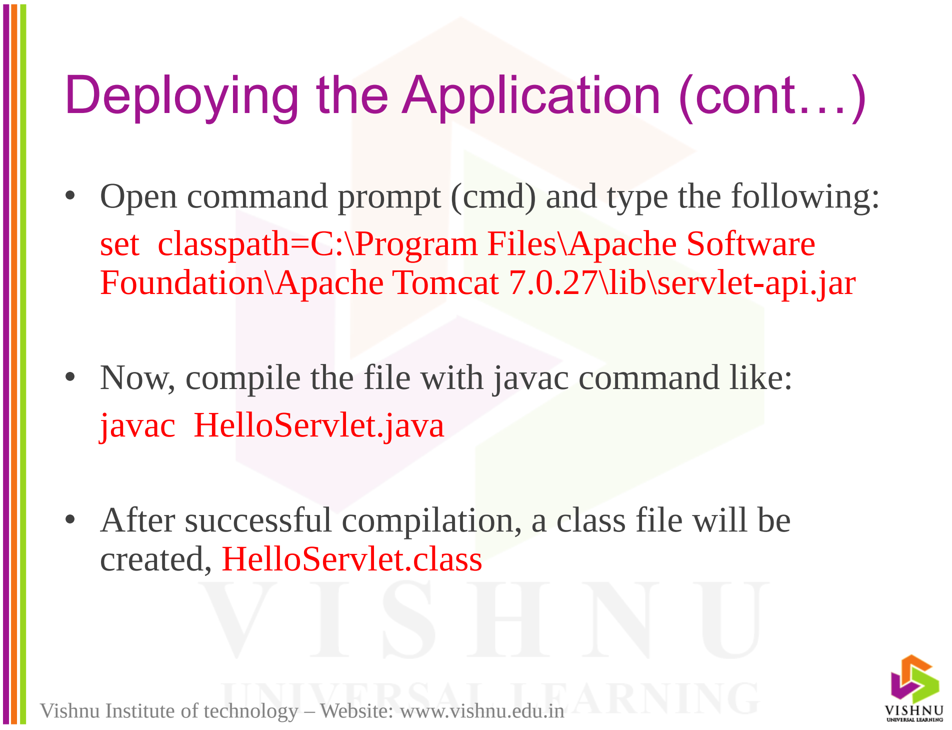
web.xml

```
<?xml version="1.0" ?>
<web-app>
    <servlet>
          <servlet-name>HelloServlet</servlet-name>
          <servlet-class>HelloServlet</servlet-class>
    </servlet>
    <servlet-mapping>
          <servlet-name>HelloServlet</servlet-name>
          <url-pattern>/HelloServlet</url-pattern>
    </servlet-mapping>
</web-app>
```



Deploying the Application

- At this point you will have three files:
  - index.html
  - HelloServlet.java
  - web.xml
- Now, compile the HelloServlet.java file.
- To compile it, we have to set the CLASSPATH environment variable.



Deploying the Application (cont…)

- Open command prompt (cmd) and type the following:

```
set  classpath=C:\Program Files\Apache Software Foundation\Apache Tomcat 7.0.27\lib\servlet-api.jar
```

- Now, compile the file with javac command like:

```
javac  HelloServlet.java
```

- After successful compilation, a class file will be created, HelloServlet.class

Deploying the Application (cont…)

- Now, arrange the 4 files according to the following directory structure:

```
<workfolder> *.html, *.jsp, *.java, *.css etc…
     |
      -------- WEB-INF
                    |
                     -----classes (.class files)
                     -----lib (.jar or .zip files if any)
                     -----web.xml file
```

Deploying the Application (cont…)

- Create a WAR file using the jar tool. Navigate to your work folder and type the following command at the command prompt:

```
jar  -cvf  helloworld.war  *
```

- A helloworld.war file will be created in the work folder.
- This step is optional.

Executing the Application

- Create a environment variable JAVA_HOME with value set to the JDK home directory. This is not need for the latest version of apache tomcat server.
- Start tomcat server.
- Copy the helloworld.war file into tomcat's webapps folder.
- Open a browser like firefox and in the address bar type: http://localhost:8080/helloworld/
- That's it! Your application is running ☺

javax.servlet.Servlet Interface

- *javax.servlet.Servlet* interface provides a standard abstraction for the servlet container to understand the servlet object created by the user.
- Five methods available in this interface are:
  - void init(ServletConfig)
  - void service(ServletRequest, ServletResponse)
  - void destroy()
  - ServletConfig getServletConfig()
  - String getServletInfo()

javax.servlet.ServletConfig Interface

- The *javax.servlet.ServletConfig* interface provides a standard abstraction for the servlet object to get environment details from the servlet container.

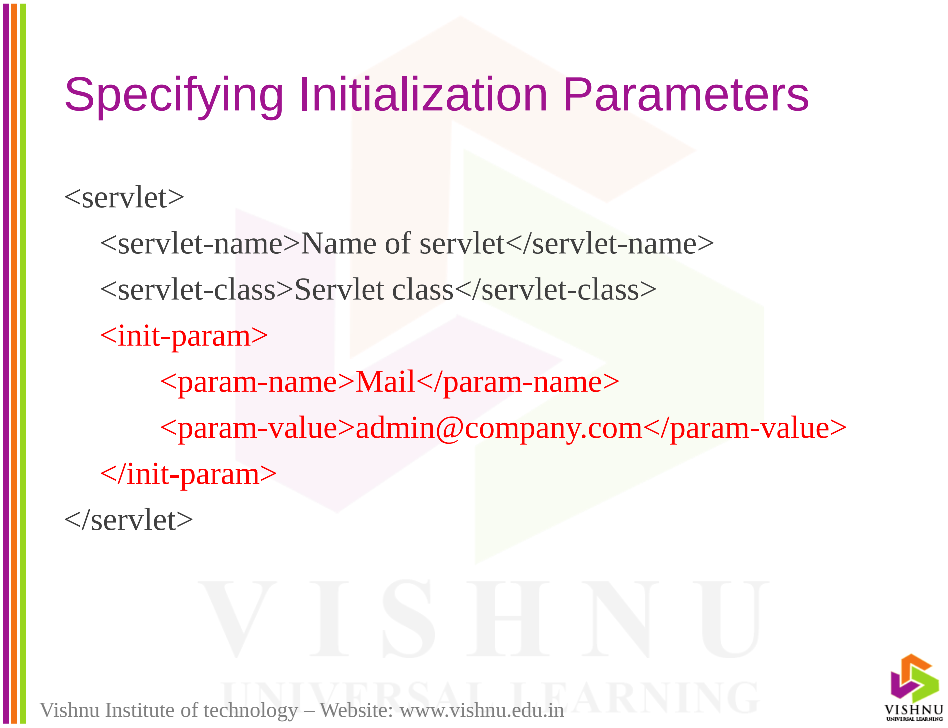
- Container implementing the *ServletConfig* object supports the following operations:
  - Retrieve the initialization parameters from web.xml file.
  - Retrieve the *ServletContext* object that describes the application's runtime environment.
  - Retrieve the servlet name as configured in web.xml file.

Initialization Parameters

- Most of the times, data (Ex: admin email, database username and password, user roles etc…) need to be provided in the production mode (client choice).

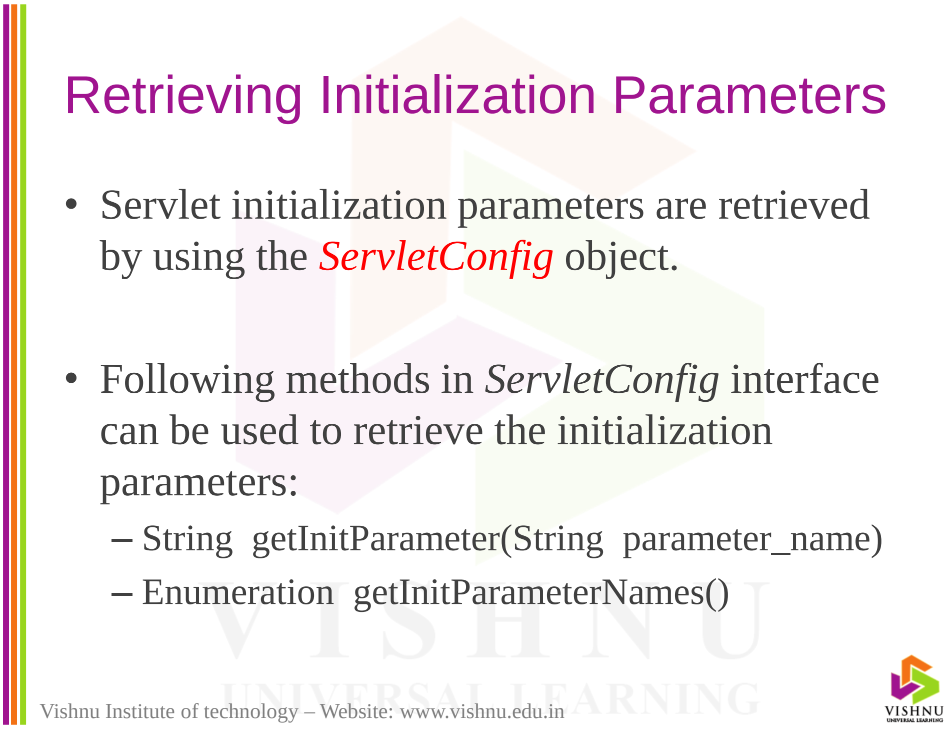
- Initialization parameters can reduce the complexity and maintenance.

- Initialization parameters are specified in the web.xml file.



Specifying Initialization Parameters

```
<servlet>
    <servlet-name>Name of servlet</servlet-name>
    <servlet-class>Servlet class</servlet-class>
    <init-param>
        <param-name>Mail</param-name>
        <param-value>admin@company.com</param-value>
    </init-param>
</servlet>
```



Retrieving Initialization Parameters

- Servlet initialization parameters are retrieved by using the *ServletConfig* object.

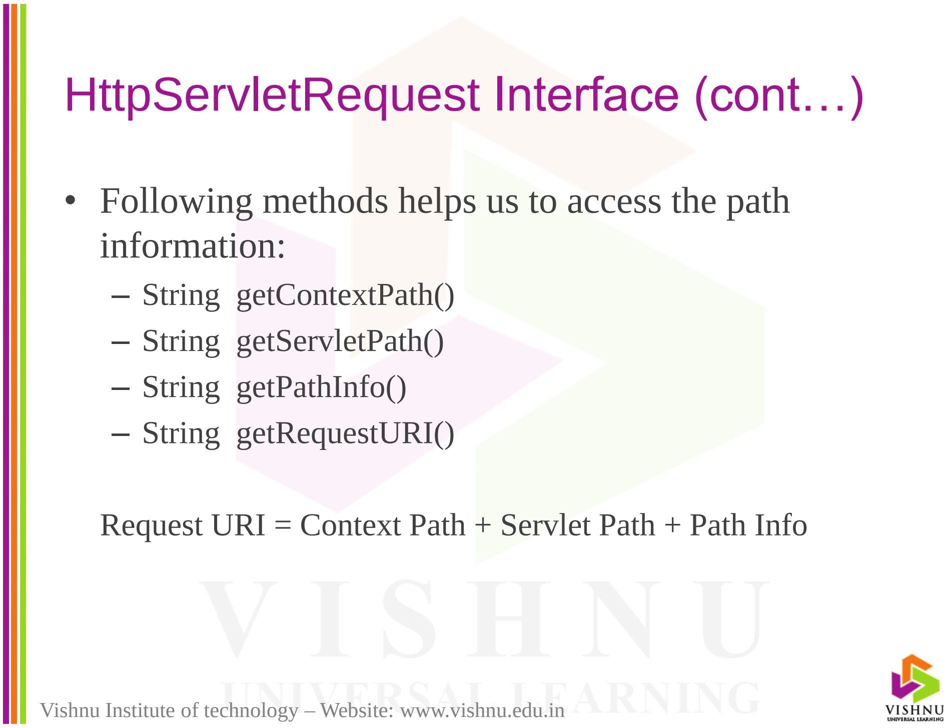
- Following methods in *ServletConfig* interface can be used to retrieve the initialization parameters:
  - String getInitParameter(String parameter_name)
  - Enumeration getInitParameterNames()

HttpServletRequest Interface

- The *HttpServletRequest* interface is a subtype of *ServletRequest* interface. Implementation for this interface is provided by the servlet container.
- The *HttpServletRequest* interface object allows us to access the data available in the HTTP headers and HTTP requests.

HttpServletRequest Interface (cont…)

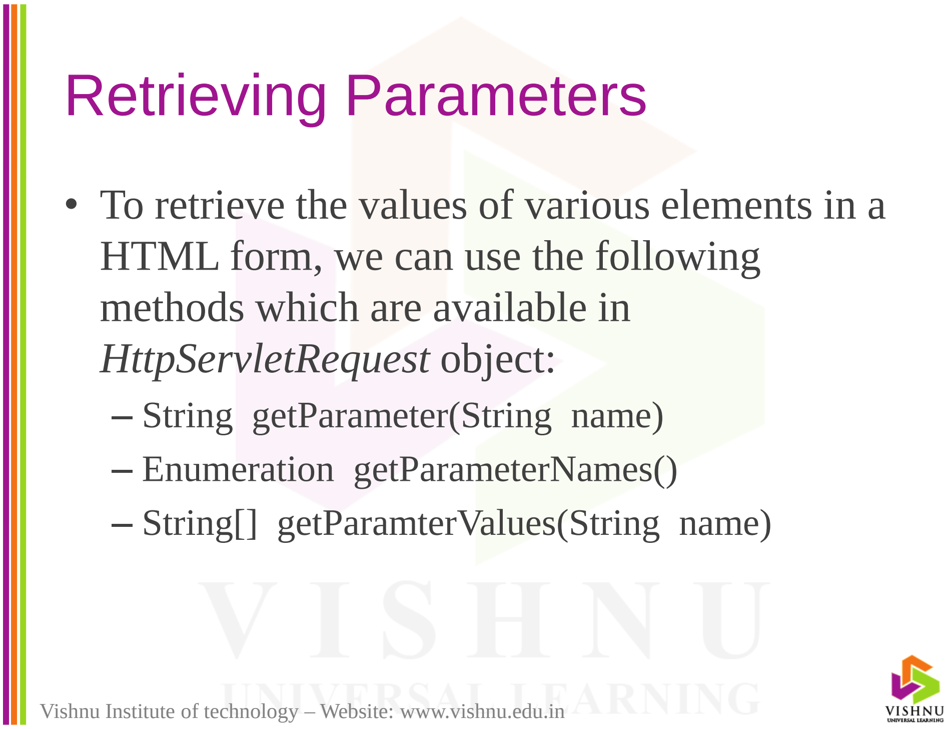
- Following methods helps us to access the header information:
  - String getHeader(String)
  - Enumeration getHeaders(String)
  - Enumeration getHeaderNames()



HttpServletRequest Interface (cont…)

- Following methods helps us to access the path information:
  - String getContextPath()
  - String getServletPath()
  - String getPathInfo()
  - String getRequestURI()

Request URI = Context Path + Servlet Path + Path Info



Retrieving Parameters

- To retrieve the values of various elements in a HTML form, we can use the following methods which are available in *HttpServletRequest* object:
  - String getParameter(String name)
  - Enumeration getParameterNames()
  - String[] getParamterValues(String name)

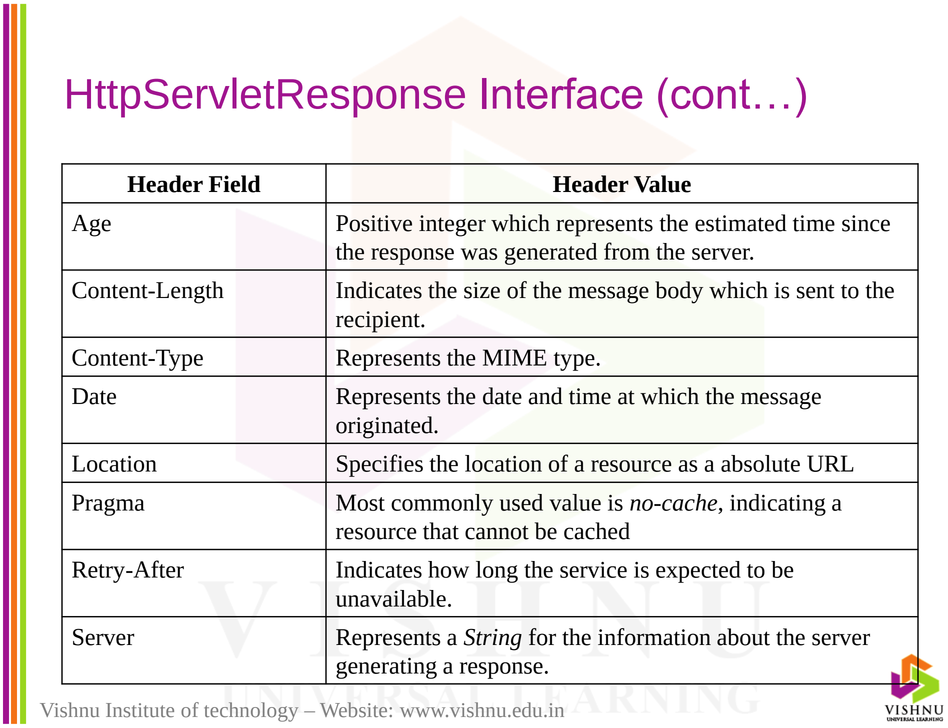
HttpServletResponse Interface

- The *HttpServletResponse* is a subtype of *ServletResponse* interface. Implementation for this interface is provided by the servlet container.

- The *HttpServletResponse* interface object let's us to specify information that will be a part of the HTTP response headers or the HTTP response.

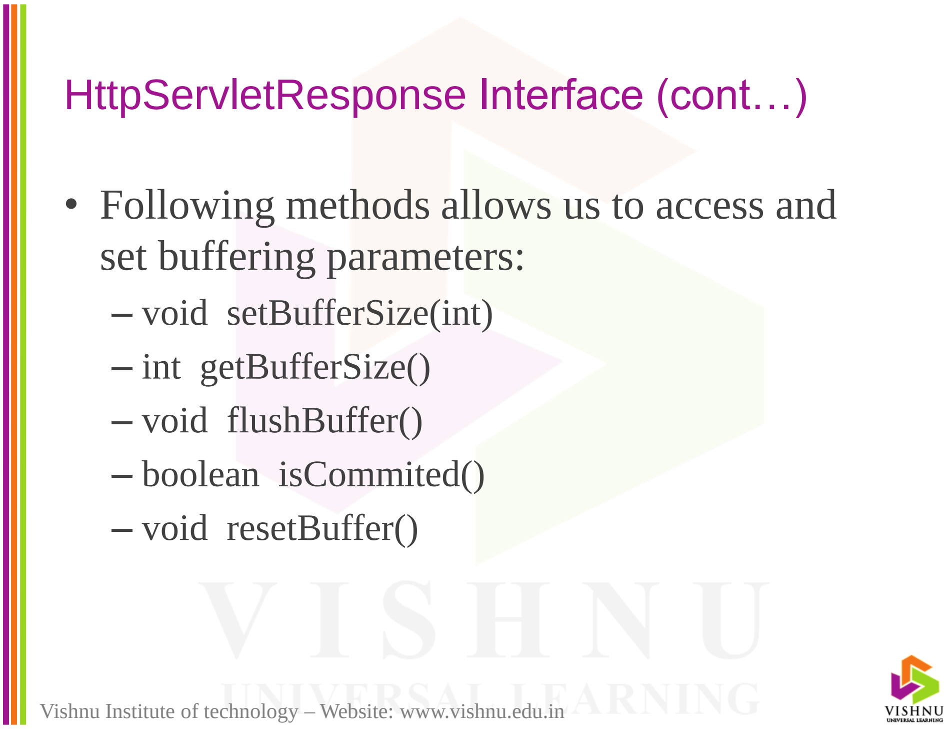
HttpServletResponse Interface (cont…)

- Following methods helps us to manipulate the HTTP response headers:
  - addHeader(String name, String value)
  - containsHeader(String name)
  - setHeader(String name, String value)
  - setDateHeader(String name, long date)
  - addIntHeader(String name, int value)
  - addDateHeader(String name, long date)



HttpServletResponse Interface (cont…)

Header Field	Header Value
Age	Positive integer which represents the estimated time since the response was generated from the server.
Content-Length	Indicates the size of the message body which is sent to the recipient.
Content-Type	Represents the MIME type.
Date	Represents the date and time at which the message originated.
Location	Specifies the location of a resource as a absolute URL
Pragma	Most commonly used value is *no-cache*, indicating a resource that cannot be cached
Retry-After	Indicates how long the service is expected to be unavailable.
Server	Represents a *String* for the information about the server generating a response.



HttpServletResponse Interface (cont…)

- Following methods allows us to access and set buffering parameters:
  - void setBufferSize(int)
  - int getBufferSize()
  - void flushBuffer()
  - boolean isCommited()
  - void resetBuffer()

HttpServletResponse Interface (cont…)

- Other methods supported:
  - sendError(int)
  - sendError(int, String)
  - sendRedirect(String URL)

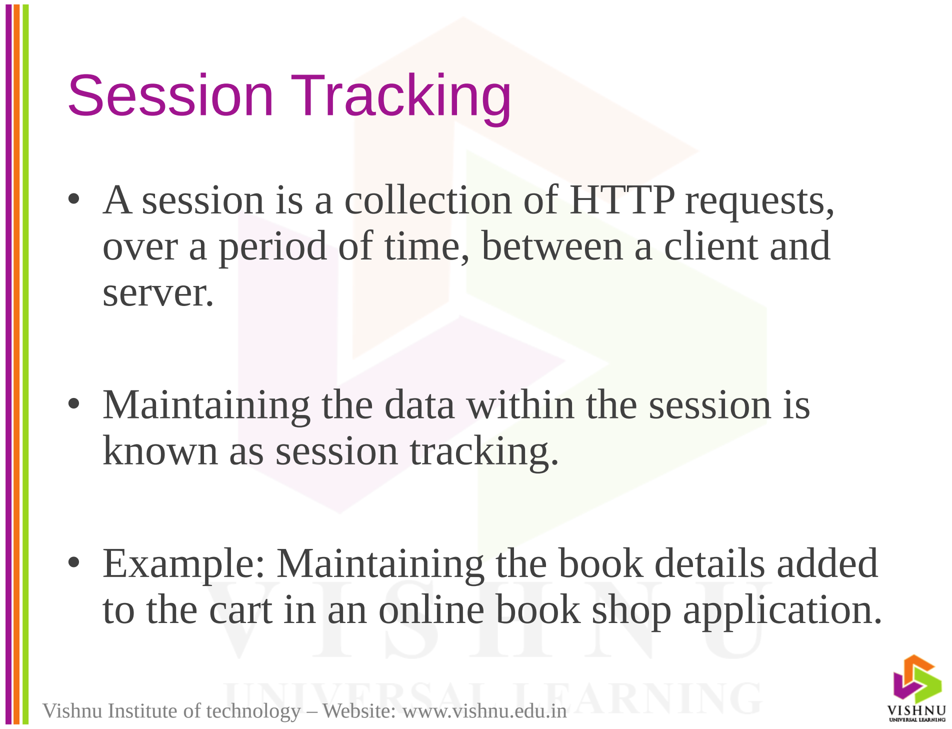
- After the *sendError()* method is invoked, we cannot send any response messages to the client

RequestDispatcher.forward() vs sendRedirect()

- *sendRedirect()* is executed on client side, whereas *forward()* is executed on server side.

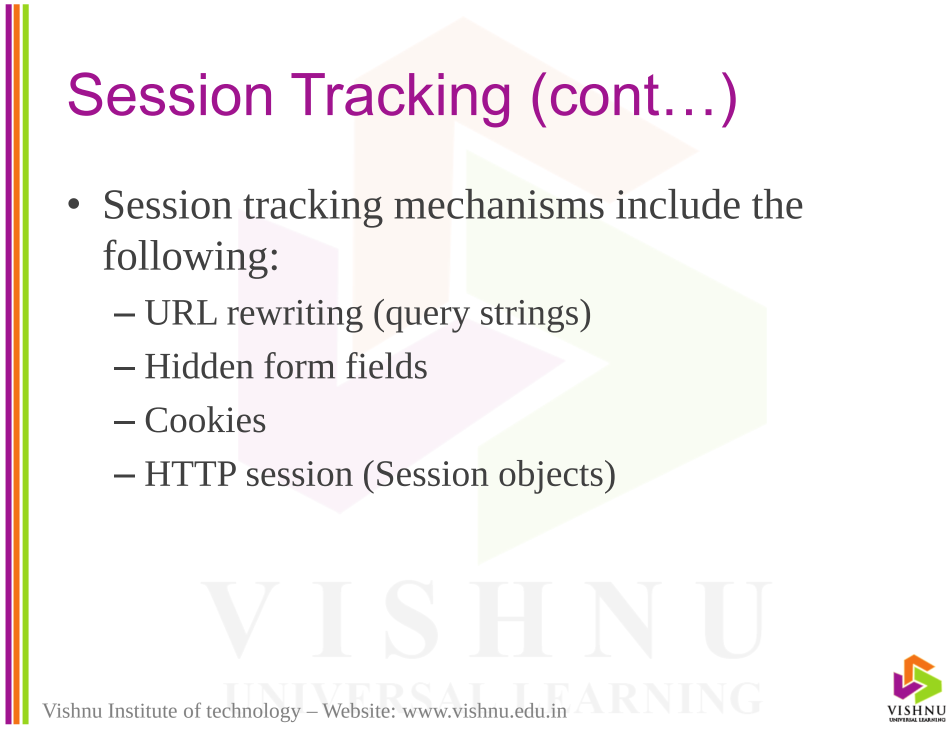
- *sendRedirect()* works only with HTTP, whereas *forward()* works with any protocol.

- *sendRedirect()* takes two request and one response to complete, where as only one call is consumed in the case of the *forward()* method.



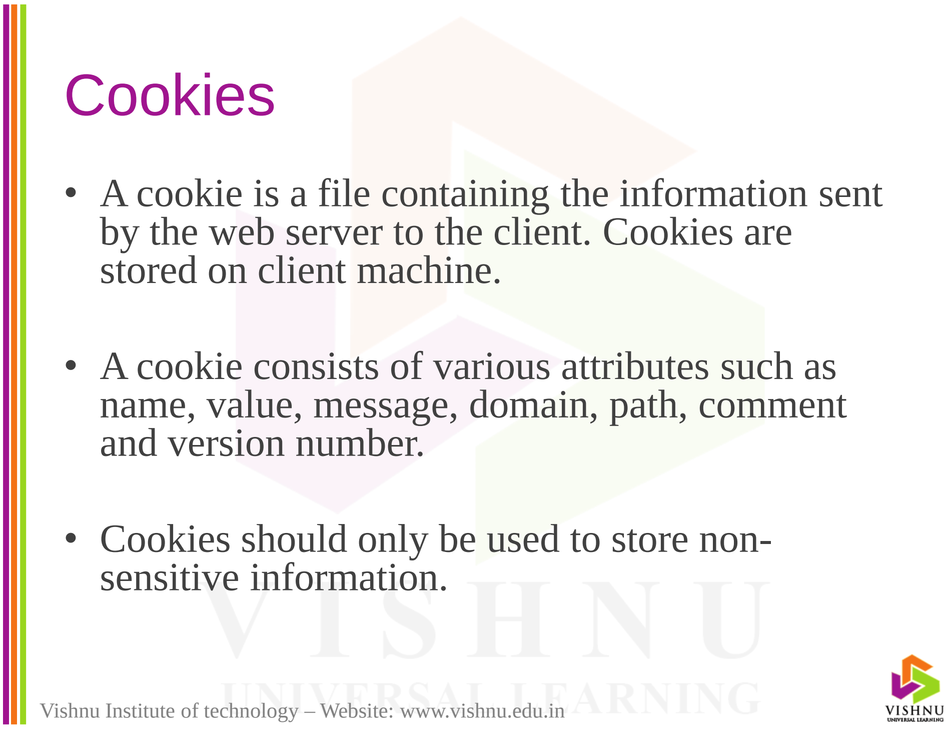
Session Tracking

- A session is a collection of HTTP requests, over a period of time, between a client and server.
- Maintaining the data within the session is known as session tracking.
- Example: Maintaining the book details added to the cart in an online book shop application.



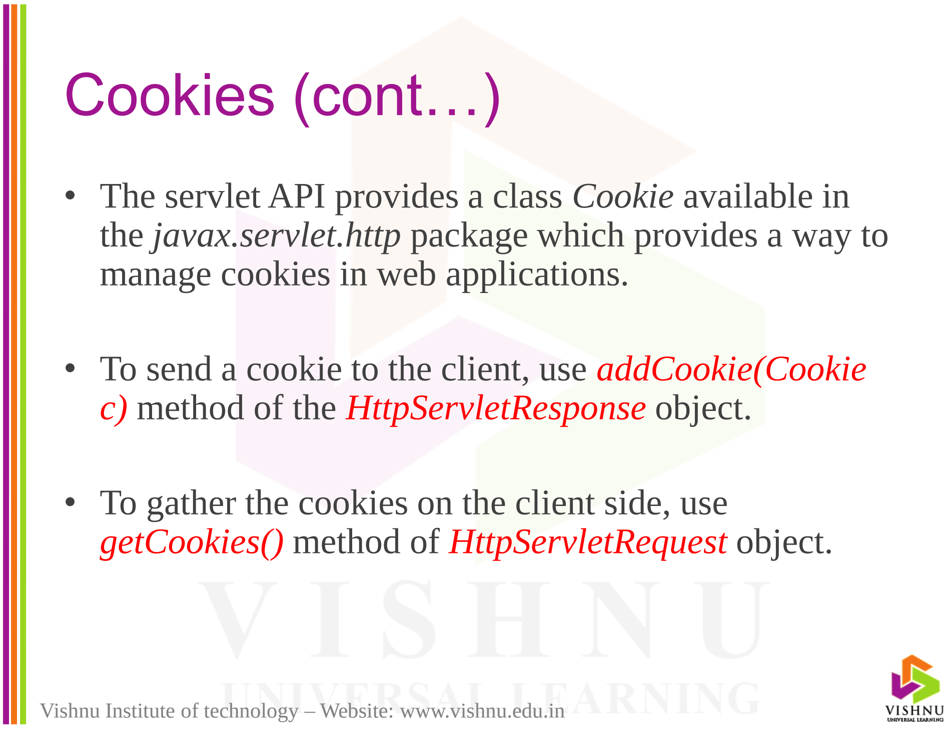
Session Tracking (cont…)

- Session tracking mechanisms include the following:
  - URL rewriting (query strings)
  - Hidden form fields
  - Cookies
  - HTTP session (Session objects)



Cookies

- A cookie is a file containing the information sent by the web server to the client. Cookies are stored on client machine.
- A cookie consists of various attributes such as name, value, message, domain, path, comment and version number.
- Cookies should only be used to store non-sensitive information.



Cookies (cont…)

- The servlet API provides a class *Cookie* available in the *javax.servlet.http* package which provides a way to manage cookies in web applications.
- To send a cookie to the client, use *addCookie(Cookie c)* method of the *HttpServletResponse* object.
- To gather the cookies on the client side, use *getCookies()* method of *HttpServletRequest* object.

Cookies (cont…)

- To create a cookie, we can use the constructor of the *Cookie* class as shown below:

  Cookie c = new Cookie(name, value);

Cookies (cont…)

- *Cookie* class provides the following methods:
  – setValue(String s)
  – getValue()
  – getName()
  – setComment(String s)
  – getComment()
  – setVersion(String s), getVersion()
  – setDomain(String s), getDomain()
  – setPath(String s), getPath()
  – setSecure(boolean), getSecure(boolean)

Advantages of Cookies

- Cookies reduce network traffic when compared to URL rewriting.
- Cookies maintain data on client side.
- Cookies simplifies the application logic when compared to query strings.

Disadvantages of Cookies

- Cookies are not secure.
- Cookies are HTTP specific.
- Cookies size is limited (4KB in general).
- Cookies can be disabled on client side.

Session Object

- Session object allows the user to store session data on the server-side. This may be a burden on the server if the application is accessed by large number of users.
- Servlet API provides *HttpSession* interface to manage the session objects.
- We can get the reference of a session object by calling *getSession()* of *HttpServletRequest* as shown below:

  HttpSession session = request.getSession();

Session Object (cont…)

- HttpSession interface provides the following functionality:
  - Object getAttribute(String name)
  - Enumeration getAttributeNames()
  - String getId()
  - void invalidate()
  - void setAttribute(String name, Object value)
  - void removeAttribute(String name)