Advanced Java Programming (CIE-306T)

Unit -2

- Preparing a Class to be a Java Bean
- Creating a Java Bean, Java Bean Properties
- Types of beans: Stateful Session bean, Stateless Session bean, Entity bean
- Servlet Overview and Architecture
- Interface Servlet and the Servlet Life Cycle
- Handling HTTP GET Requests
- Handling HTTP POST Requests
- Session Tracking, Cookies

JAVA Bean

A JavaBean is a Java class that should follow the following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use JavaBean?

According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides easy maintenance.

Simple example of JavaBean class (Employee.java)

```
package mypack;
public class Employee implements java.io.Serializable{
    private int id;
    private String name;
    public Employee(){}
    public void setId(int id){
        this.id=id;
    }
    public int getId(){
        return id;
    }
    public void setName(String name){
        this.name=name;
    }
    public String getName(){
        return name;
    }
}
```

How to access the JavaBean class?

To access the JavaBean class, we should use getter and setter methods. package mypack;

```
public classTest{
    public static void main(String args[]){
        Employee e = new Employee();
        e.setName("Arjun");
        System.out.println(e.getName());
    }
}
```

JavaBean Properties

A JavaBean property is a named feature that can be accessed by the user of the object.The feature can be of any Java data type, containing the classes that you define.

A JavaBean property may be read, write, read-only, or write-only. JavaBean features are accessed through two methods in the JavaBean's implementation class:

1. getPropertyName ()

For example, if the property name is firstName, the method name would be getFirstName() to read that property.This method is called the accessor.

2. setPropertyName ()

For example, if the property name is firstName, the method name would be
setFirstName() to write that property.This method is called the mutator.

Advantages of JavaBean

The following are the advantages of JavaBean:

The JavaBean properties and methods can be exposed to another application.

It provides an easiness to reuse the software components.

Disadvantages of JavaBean

The following are the disadvantages of JavaBean:

JavaBeans are mutable. So, it can't take advantages of immutable objects.

Creating the setter and getter method for each property separately may lead to the boilerplate code.

JavaBean class in Java

JavaBeans are classes that encapsulate many objects into a single object (the bean). It is a Java class that should follow the following conventions:

Must implement Serializable.

It should have a public no-arg constructor.

All properties in Java Bean must be private, using public getters and setter methods.

Illustration of JavaBean Class

A simple example of JavaBean Class is mentioned below:

```
/ /  Java program to illustrate the structure of JavaBean


class public class TestBean {
        private String name;
        public void setName(String name){
                this.name =  name;
                }
                public String getName() {
                        return name;
                }
}
```

Setter and Getter Methods in Java

Setter and Getter Methods in Java properties are mentioned below:

Properties for setter methods:

- It should be public in nature.
- The return type **a** should be void.
- The setter method should be prefixed with the set.
- It should take some argument i.e. it should not be a no-arg method.

Properties for getter methods:

- It should be public in nature.
- The return type should not be void i.e. according to our requirement, **return type** we have to give the return type.
- The getter method should be prefixed with get.
- It should not take any argument.
- For Boolean properties getter method name can be prefixed with either "get" or "is". But recommended to use "is".

```
// Java program to illustrate the
// getName() method on boolean type attribute

public class Test {
        private boolean empty;
        public boolean getName(){
        return empty;
        }
        public boolean isempty(){
        return empty;
        }
}
```

```
// Java Program of JavaBean class package geeks;

public class Student implements java.io.Serializable {
                private int id;
                private String name;
                // Constructor
                public Student() {}
                // Setter for Id
                public void setId(int id) { this.id = id; }
                // Getter for Id
                public int getId() {
                        return id;
                }
                // Setter for Name
                public void setName(String name) {
                        this.name = name;
                }
                // Getter for Name
                public String getName() { return name; }
}
```

```
// Java program to access JavaBean class
package geeks;

// Driver Class
public class Test {
        // main function
        public static void main(String args[])
        {
                // object is created
                Student s = new Student();

                // setting value to the object
                s.setName("GFG");

                System.out.println(s.getName());
        }
}
```

Enterprise Java Beans

- EJB is an acronym for enterprise java bean. It is a specification provided by Sun Microsystems to develop secured, robust and scalable distributed applications.
- To get information about distributed applications, visit RMITutorial first.
- To run EJB application, you need an application server (EJB Container) such as Jboss, Glassfish,Weblogic,Websphere etc. It performs:
    a. life cycle management,
    b. security,
    c. transaction management, and
    d. object pooling.
- EJB application is deployed on the server, so it is called server side component also.
- EJB is like COM (Component Object Model) provided by Microsoft. But, it is different from Java Bean, RMI andWeb Services.

When use Enterprise Java Bean?

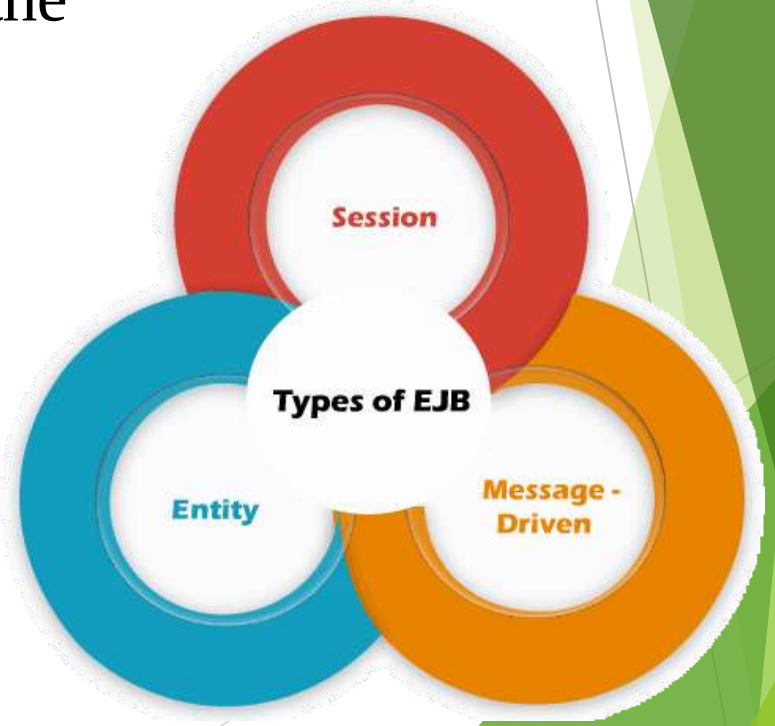
- Application needs Remote Access. In other words, it is distributed.
- Application needs to be scalable. EJB applications supports load balancing, clustering and fail-over.
- Application needs encapsulated business logic. EJB application is separated from presentation and persistent layer

Types of EJB

EJB is an acronym for Enterprise Java Beans. It is a server-side software element. It encapsulates the business logic of an application. It is a specification for developing a distributed business application on the Java platform.

There are three types of EJBs:

- Session Bean
- Entity Bean
- Message-Driven Bean



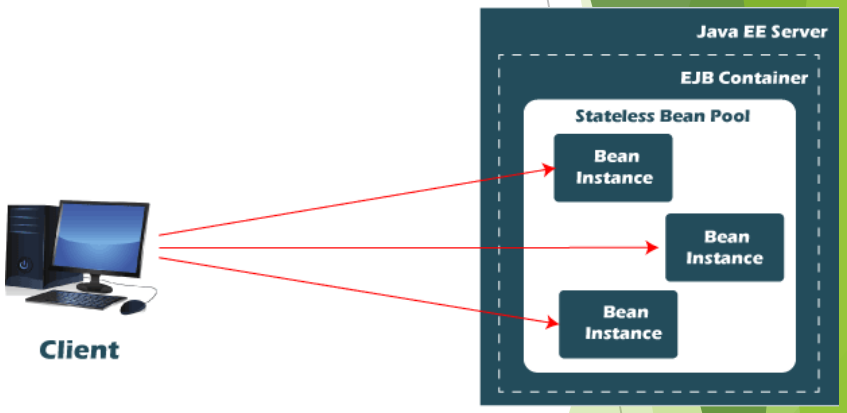
Session Bean

- The bean sprout business idea that can be systematically used by customer, remote, and web service.When a client wants to use an application distributed on a server, then the client uses session bean methods.This session bean provides services to customers, protecting them from complex problems by performing business operations within the server.
- To get to an application that is conveyed to the worker, the client summons the meeting bean's strategies.The meeting bean performs work for its client, safeguarding it from intricacy by executing business errands inside the worker. Remember that session beans are not persistence.

- There are two types of session beans:
- Stateless Session Beans
- Stateful Session Beans
- Singleton Session Beans

Stateless Session Beans

- A **stateless session bean** doesn't keep a conversational state with the customer or client. When a client invokes the methods for a stateless bean, the bean's instance variable may contain a state explicitly to that client however only for the span of the invocation.
- At the point when the method has finished, the client explicit state should not be held. However, the client may change the state of instance variable presented in pooled stateless beans, and this state is held over to the following invocation of the pooled stateless bean.

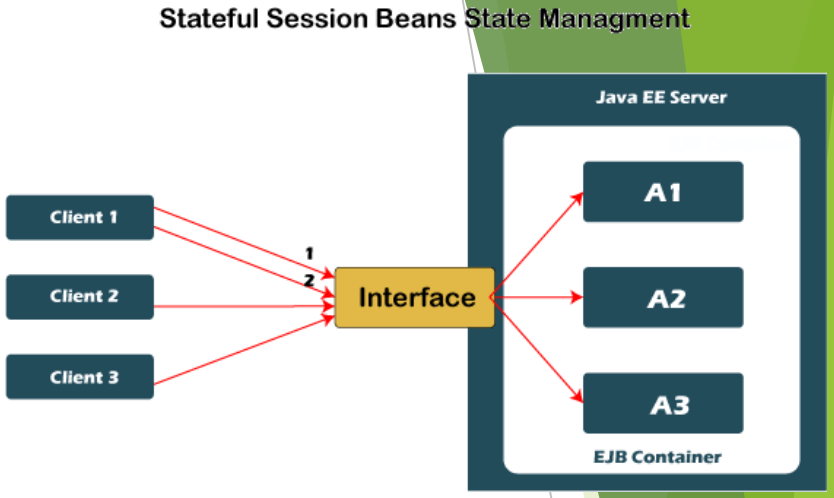


Stateless Session Beans

- Besides this, during the invocation of the method, all instances of a stateless bean are the same, permitting the EJB container to assign an instance to any client. Therefore, the state of a stateless session bean should apply across all clients. It can be implemented in a web-services.
- Since they can maintain various customers, stateless session beans can offer better adaptability for applications requiring large clients. Ordinarily, an application requires fewer stateless session beans than stateful session beans to support the same number of clients.
- To improve execution, we may choose a stateless session bean if it has any of these characteristics.
- The state of the bean has no information or data for a particular client.
- In a private method invocation, the bean performs a generic task for all clients.
- The bean implements a web service.

Stateful Session Beans

- A **stateful session bean** is the same as an interactive session. The state of an object comprises the values of its instance variables. In a stateful session bean, the instance variables denote the state of a unique client/bean session. Since, the client interacts with its bean. The state is usually called the **conversational state**. It maintains the state of a client across multiple requests. So, we cannot implement it in the web services because it cannot be shared. When the client terminates the session, it is no longer associated with the client.
- The state is held for the span of the client/bean session. In case if the client eliminates the bean, the session closes and the state is omitted. This transient nature of the state isn't an issue, because the interaction between the client and the beans ends. So, it is not required to hold the state.



Stateful session beans should be used, if any of the following conditions are valid:

- The bean's state denotes the alliance between the bean and a particular client.
- The bean needs to hold data about the customer across method invocations.
- The bean interferes between the customer and different segments of the application, introducing an improved visible to the client.
- In the background, the bean deals with the workstream of a few enterprise beans.

Note: Stateless and stateful both session beans are not persistent.

Singleton Session Beans

- A **singleton session bean** maintains one instance per application and the instance exists for the lifecycle of the application. It offers the same functionality as the stateless session beans. But the only difference is that there is only one singleton session bean per application while in the stateless session bean a pool of beans is used.
- From that pool, any of the session beans may respond to the client. We can implement it in the web-service endpoints. It takes care of the state but does not hold the state if unexpected crashes or shutdown occur. It can be used if we want to perform cleanup tasks on closing the application or shut down as well. It is because it operates throughout the life cycle of the application.

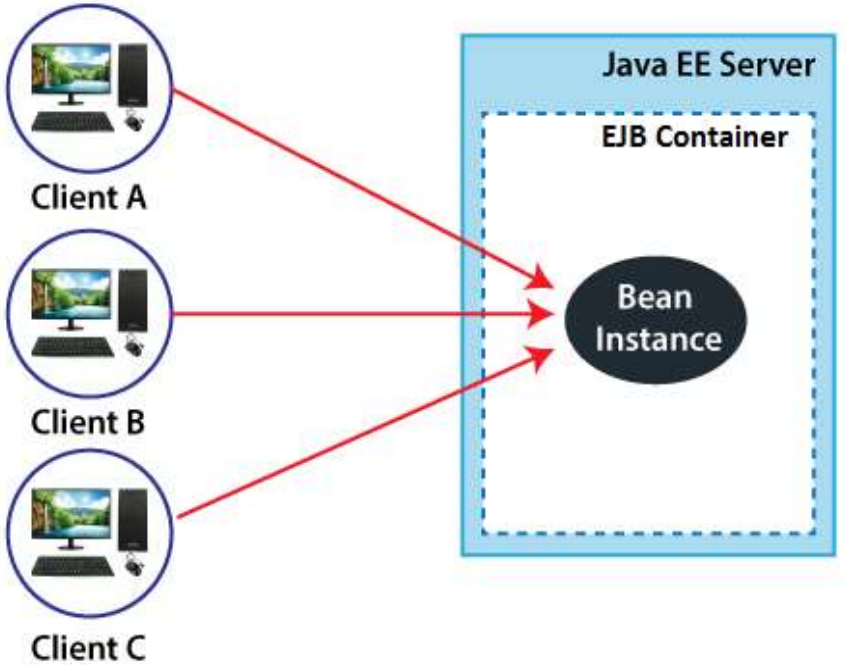


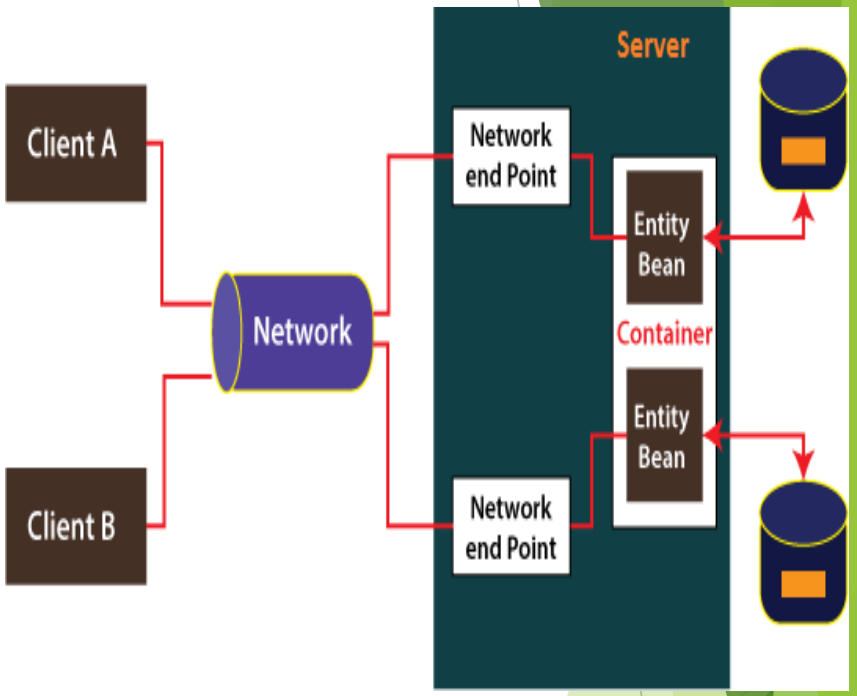
Fig: Singleton Session Bean

Singleton session beans are suitable for the following conditions:

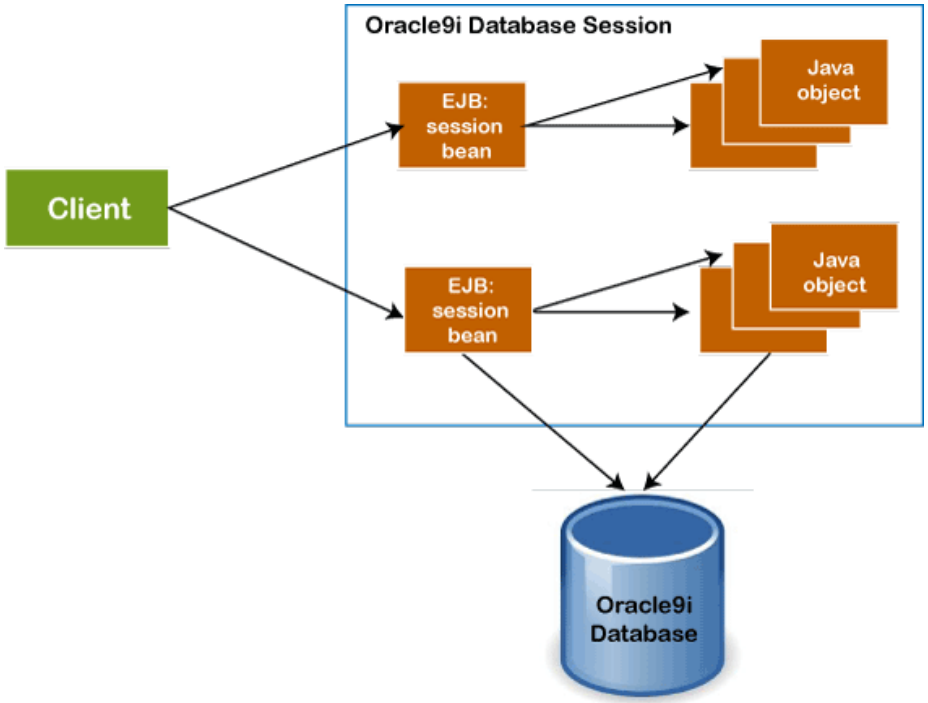
- The state of the bean must be shared across the application.
- A single enterprise bean should be accessed by multiple threads concurrently.
- The application needs an enterprise bean to perform undertakings upon application startup and shutdown.
- The bean implements a web service.

Entity Bean

- activities inside a business interaction. It is used to encourage business benefits that include data and calculations on that data. It can deal with various, needy, diligent articles in playing out its essential assignments. A substance bean is a far-off object that oversees steady information and performs complex business rationale. It can be extraordinarily distinguished by an essential key.
- It is a far-off object that oversees steady information, performs complex business rationale, possibly utilizes a few ward Java protests, and can be remarkably distinguished by an essential key. It ordinarily coarse-grained determined items since they use steady information put away inside a few fine-grained relentless Java objects. Element beans are diligent on the grounds that they do endure a worker crash or an organization's disappointment.



Entity Bean contd.



Every entity bean has a persistence identity that is associated with it. It means that it comprises a unique identity that can be fetched if we have a primary key. The type for the unique key is defined by the bean provider. A client can retrieve the entity bean if the primary has been misplaced. If the bean is not available, the EJB container first, instantiates the bean and then re-populates the data for the client.

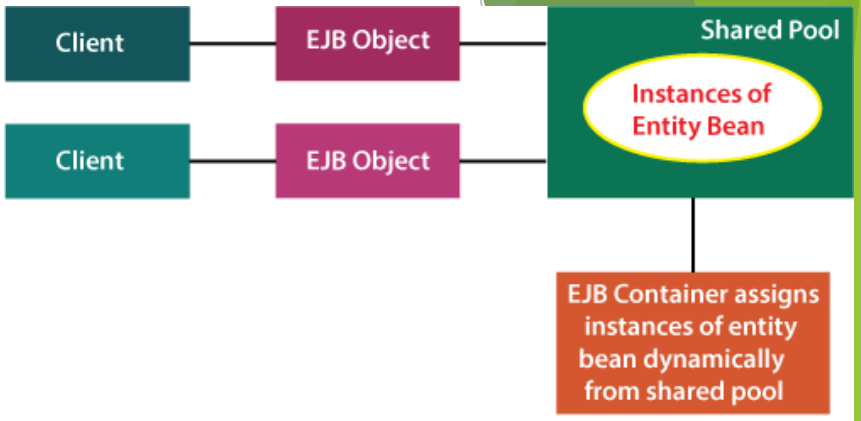


Fig: Pooling and Instances of Entity Bean

The diligence for entity bean information is given both to saving state when the bean is passivated and for improving the state when a failover has detected. It can endure in light of the fact that the information is put away determinedly by the holder in some type of information stockpiling framework, like a data set.

Entity beans persist business data by using the following two methods:

- Bean-Managed Persistence (BMP)
- Container-Managed Persistence (CMP)

- **Bean Managed Persistence**
- Bean-managed persistence is more complicated in comparison to container-managed persistence. Because it allows us to write the persistence logic into the bean class, explicitly. In order to write the persistence handling code into the bean class, we must know the type of database that is being used and how fields of the bean class are mapped to the database. Therefore, it provides more flexibility between the database and the bean instance.
- It can be used as an alternative to CMP. If the deployment tools are incompatible for mapping the bean instance's state to the database. The disadvantage of BMP is that more work is required to define the bean and it ties the bean to a specific database type and structure.

- **Container-Managed Persistence**
- In **container-managed persistence,** the EJB container transparently and implicitly handles the relationship between the bean and the database. Bean developers focus on the data and the business process. The principal constraint is that the EJB compartment can most likely not produce information base access articulations with the proficiency of a programmer.
- Unlike BMP, CMP does not allow us to write database access calls in the methods of the entity bean class. It is because the persistence is handled by the container at run-time. The following two things are required to support the CMP:
- **Mapping:** It denotes that how to map an entity bean to a resource.
- **Runtime Environment:** A CMP runtime environment that uses the mapping information to perform persistence operations on each bean.

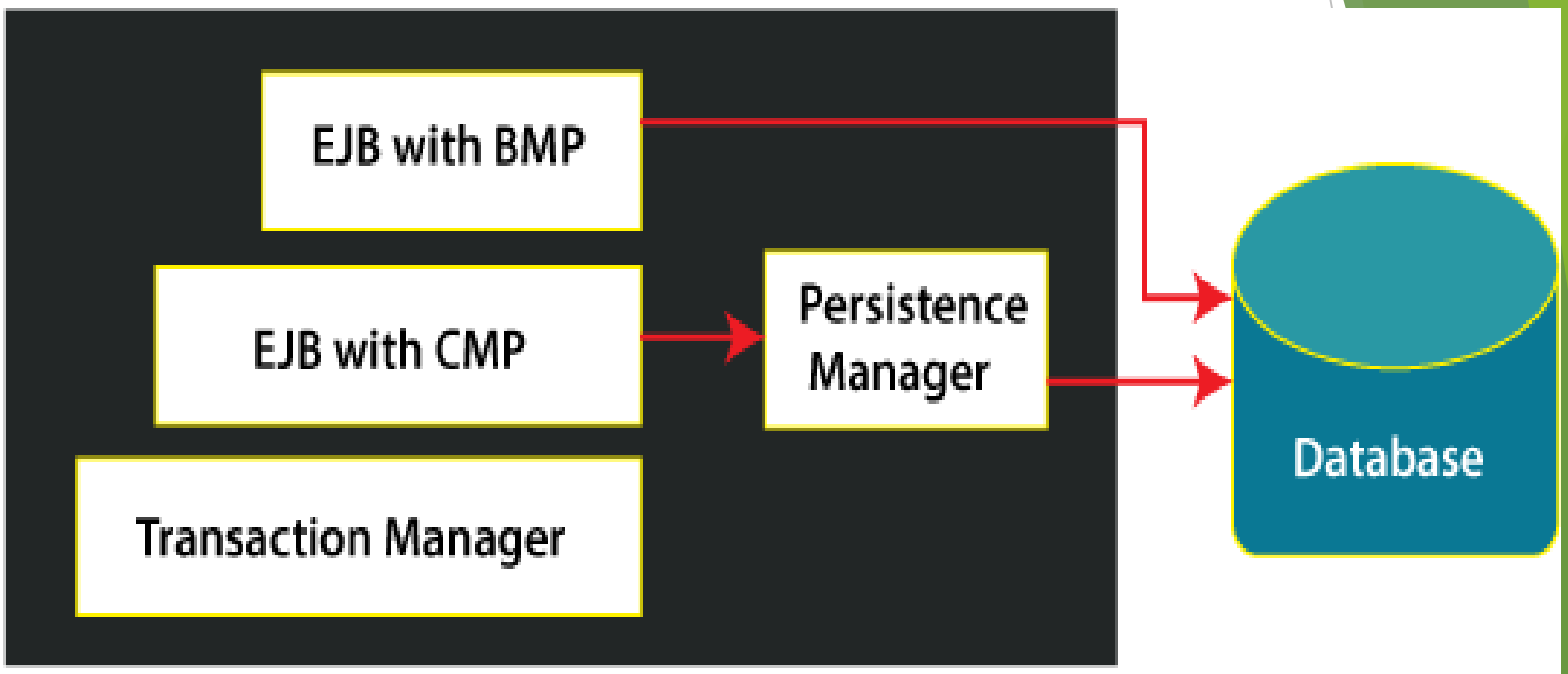
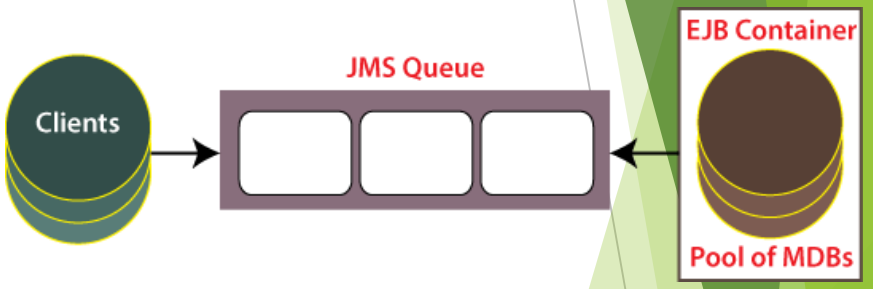


Fig: Entity Bean Flow

- ***Note: Entity bean has been replaced with Java persistence API.***

Message-Driven Bean (MDB)

- **MDB** is a **Java Messaging Service** (message listener). It consumes messages from a queue or subscription of a topic. It provides an easy way to create or implement asynchronous communication using a JMS message listener. An advantage of using MDB is that it allows us to use the asynchronous nature of a JMS listener. We can implement message-driven beans with Oracle JMS (Java Messaging Service).
- There is an EJB container (contains MDBs pool) that handles the JMS queue and topic. Each incoming message is handled by a bean that is invoked by the container. Note that no object invokes an MDB directly. All invocation for MDB originates from the container. When the container invokes the MDB, it can invoke other EJBs or Java objects to continue the request.



- It is quite similar to stateless session bean. Because it does not save informal state, also used to handle multiple incoming requests. EJB has the following benefits over the JMS are as follows:
- It creates a consumer for the listener. It means that the container creates **QueueReceiver** or **TopicSubscriber** is created by the container.
- MDB is registered with the consumer. It means that at deployment time QueueReceiver,TopicSubscriber, and its factory are registered by the container.
- The message ACK mode is specified.
- The primary function of MDB is to read (receive) or write (send) incoming from a JMS destination (i.e. queue or topic). While using MDB ensure that it is configured and installed properly. It interacts with JMS and the JMS installed on the Oracle database. The database retains queue or topic. The following figure depicts how MDB interacts with the JMS destination.

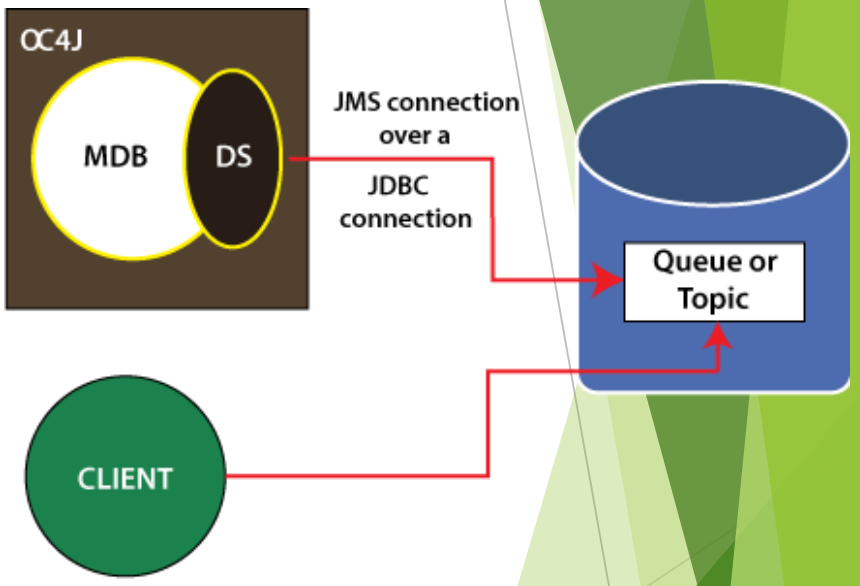


Fig: MDB Interacting with JMS Destination

working of MDB

The working of MDB is as follows:

- MDB creates and opens a JMS connection to the database by using the data source (JMS resource provider).The JDBC driver is used to facilitate the JMS connection.
- A JMS session over the JMS connection is opened by the MDB.
- If any message for the MDB is routed to the onMessage() method of the MDB from the queue o topic. Other clients may also access the same queue and topic to put the message for the MDB.
- Note that it does not handle the requests that are requested by the clients instead it handles requests that are placed into a queue.

- MDB implements the javax.ejb.MessageDriverBean interface and inherits the javax.jms.MessageListener class that provides the following methods:

Method	Description
onMessage(msg)	The container dequeues a message from the JMS queue associated with this MDB and gives it to this instance by invoking this method. This method must have an implementation for handling the message appropriately.
setMessageDrivenContext (ctx)	After the bean is created, the setMessageDrivenContext method is invoked. This method is similar to the EJB setSessionContext and setEntityContext methods.
ejbCreate()	This method is used just like the stateless session bean ejbCreate method. No initialization should be done in this method. However, any resources that you allocate within this method will exist for this object.
ejbRemove()	Delete any resources allocated within the ejbCreate method.

Difference Between Session and Entity Bean

Basis of Comparison	Session Bean	Entity Bean
Primary Key	It has no primary key. It is used to identify and retrieve specific bean instances.	It has a primary key that allows us to find instances and can be shared by more than one client.
Stateless/ Stateful	It may be stateful or stateless.	It is stateful.
Span	It is relatively short-lived.	It is relatively long-lived.
Performance	It represents a single conversation with a client.	It encapsulates persistence business data.
Accessibility	It is created and used by a single client.	It may be shared by multiple clients.
Recovery	It is not recoverable in case if EJB server fails. So, it may be destroyed.	It is recoverable if any failure has occurred.
Persistence of Data	It persists data only for the span of the conversation with the client.	Persistence beyond the life of a client instance. Persistence can be container-managed or bean-managed.
Remote Interface	It extends javax.ejb.EJBObject.	It extends javax.ejb.EJBObject.
Home Interface	It extends javax.ejb.EJBHome.	It extends javax.ejb.EJBHome.
Bean Class	It extends javax.ejb.EJBSessionBean.	It extends javax.ejb.EJBEntityBean.

Servlets

Servlets are grouped under the Advanced Java tree that is used to create dynamic web applications. Servlets are robust, well scalable, and are primarily used in developing server-side applications. If we go a little back in time, we would be able to witness that before the introduction of servlets, CGI (Common Gateway Interface) was used. Among several indigenous tasks that a servlet is capable of doing, dynamically performing client requests and responses are most common. Other tasks that a servlet can do effectively are:

- Can easily manage/control the application flow.
- Suitable to implement business logic.
- Can effectively balance the load on the server side.
- Easily generate dynamic web content.
- Handle HTTP Request and Response
- Also act as an interceptor or filter for a specific group of requests.

Main Points

- Servlets are used to develop dynamic web applications.
- Servlets are nothing but the Java programs which reside on the server side and their main purpose is to serve the client request.
- Servlets are fully compatible with Java.You can use any of the available Java APIs like JDBC inside the servlets.
- As servlets are written in Java, they are platform independent, robust, and secured.
- In Servlets, a thread is created for each request unlike in CGI where a process is created for each request. Hence, servlets give better performance than CGI.
- Servlets are protocol independent. i.e. they support FTP, SMTP, HTTP etc. protocols.

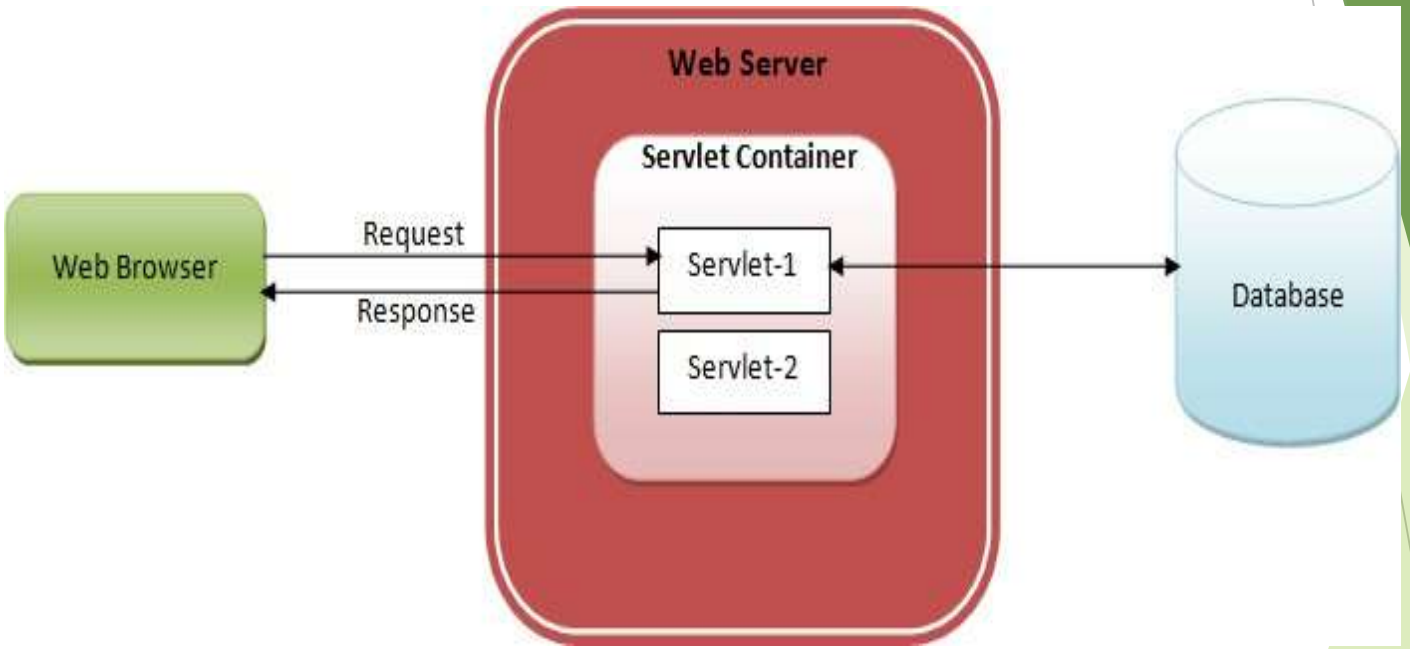
Types of Servlet

- **Generic Servlets**:These are those servlets that provide functionality for implementing a servlet. It is a generic class from which all the customizable servlets are derived. It is protocol-independent and provides support for HTTP, FTP, and SMTP protocols.The class used is '**javax.servlet.Servlet'** and it only has 2 methods – init() to initialize & allocate memory to the servlet and destroy() to deallocate the servlet.
- **HTTP Servlets**:These are protocol dependent servlets, that provides support for HTTP request and response. It is typically used to create web apps. And has two of the most used methods – doGET() and doPOST() each serving their own purpose.

There are three potential ways in which we can employ to create a servlet:

- Implementing Servlet Interface
- Extending Generic Servlet
- Extending HTTP Servlet

Components of Servlet Architecture



- Step 1 : Client i.e. web browser sends the request to the web server.
- Step 2 :Web server receives the request and sends it to the servlet container. Servlet container is also called web container or servlet engine. It is responsible for handling the life of a servlet.
- Step 3 : Servlet container understands the request's URL and calls the particular servlet. Actually, it creates a thread for execution of that servlet. If there are multiple requests for the same servlet, then for each request, one thread will be created.
- Step 4 : Servlet processes the request object and prepares response object after interacting with the database or performing any other operations and sends the response object back to the web server.
- Step 5 :Then web server sends the response back to the client.

Servlet Architecture contains the business logic to process all the requests made by client. Below is the high-level architecture diagram of servlet. Let's see in brief, how does each component add to the working of a servlet.

1. Client

- The client shown in the architecture above is the web browser and it primarily works as a medium that sends out HTTP requests over to the web server and the web server generates a response based on some processing in the servlet and the client further processes the response.

2. Web Server

- Primary job of a web server is to process the requests and responses that a user sends over time and maintain how a web user would be able to access the files that has been hosted over the server.The server we are talking about here is a software which manages access to a centralized resource or service in a network.There are precisely two types of webservers:
- Static web server
- Dynamic web server

3.Web Container

- Web container is another typical component in servlet architecture which is responsible for communicating with the servlets.Two prime tasks of a web container are:
- Managing the servlet lifecycle
- URL mapping
- Web container sits at the server-side managing and handling all the requests that are coming in either from the servlets or from some JSP pages or potentially any other file system.

How does a Servlet Request flow?

Every servlet should override the following 3 methods namely:

init():To initalize/instantiate the servlet container.

service():This method acts like an intermediatory between the HTTP request and the business logic to serve that particular request.

destroy():This method is used to deallocate the memory allocated to the servlet.

These methods are used to process the request from the user.

Following are the steps in which a request flows through a servlet which can be observed in the architecture diagram:

- The client sends over a request.
- The request is accepted by the web server and forwarded to the web container.
- In order to obtain the servlet's address, the web container traces web.xml file corresponding to the request URL pattern.
- By the time above process takes place, the servlet should have been instantiated and initialized.The init() method is invoked to initialize the servlet.
- By passing ServletRequest and Response object, public service() method is called by the container.
- In the next step, the ServletRequest and ServletResponse objects are type casted to HttpServletRequest and HttpServletResponse objects by the public service() method.
- Now protected service() method is called by the public service() method.
- The protected service() method dispatches the request to the correct handler method based on the type of request.
- When servlet container shuts down, it unloads all the servlets and calls destroy() method for each initialized servlet.

Advantages

- Prime functionality of a servlet is that they are independent of server configuration, and they are pretty much compatible with any of the web servers.
- Servlets are also protocol-independent supporting FTP, HTTP, SMTP, etc. protocols to the fullest.
- Until destroyed manually, servlets can be retained in the memory helping process several requests over time. Also, once a database connection is established, it can facilitate process several requests for a database in the very same database session.
- Servlets inherit Java's property of portability and hence are compatible with nearly any web server.
- Servlets are first converted into byte codes and then executed, which helps in increasing the processing time.

Disadvantages

- Designing a servlet can be pretty laborious.
- Exceptions need to be handled while designing a servlet since they are not thread safe.
- Developers may need additional skills to program a servlet.
- As we already know Servlets are portable (platform/server independent) in nature and hence are a better option if we talk in terms of other scripting languages. They process the requests and responses dynamically. Whenever we are developing a web application where we need to coordinate with different existing protocols, servlets are preferred over other means because of its capability to support various protocols. At last, we can descend to a conclusion that employing a servlet could potentially be most suitable while developing a web application.

Servlet Interface

Methods of Servlet interface

- **Servlet interface provides** common behavior to all the servlets. Servlet interface defines methods that all servlets must implement.
- Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

There are 5 methods in Servlet interface.The init, service and destroy are the life cycle methods of servlet.These are invoked by the web container.

Methods

Method	Description
public void init(ServletConfig config)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequest request,ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

Servlet Example by implementing Servlet interface

```
import java.io.*; import javax.servlet.*;

public class First implements Servlet{
        ServletConfig config=null;
        public void init(ServletConfig config){
        this.config=config;
        System.out.println("servlet is initialized");
        }
public void service(ServletRequest req,ServletResponse res) throws
IOException,ServletException{

        res.setContentType("text/html");

        PrintWriter out=res.getWriter();
        out.print("<html><body>");
        out.print("<b>hello simple servlet</b>");
        out.print("</body></html>");

}
public void destroy(){System.out.println("servlet is destroyed");}
public ServletConfig getServletConfig(){return config;}
public String getServletInfo(){return "copyright 2007-1010";}


}
```

GenericServlet class

- **GenericServlet** class implements **Servlet**, **ServletConfig** and **Serializable** interfaces. It provides the implementation of all the methods of these interfaces except the service method.
- GenericServlet class can handle any type of request so it is protocol-independent.
- You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Methods of GenericServlet class

There are many methods in GenericServlet class.They are as follows:

- **public void init(ServletConfig config)** is used to initialize the servlet.
- **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
- **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
- **public ServletConfig getServletConfig()** returns the object of ServletConfig.
- **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
- **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
- **public ServletContext getServletContext()** returns the object of ServletContext.
- **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
- **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
- **public String getServletName()** returns the name of the servlet object.
- **public void log(String msg)** writes the given message in the servlet log file.
- **public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

Servlet Example by inheriting the GenericServlet class

```
import java.io.*;
import javax.servlet.*;
public class First extends GenericServlet{
    public void service(ServletRequest req,ServletResponse res) throws IOException,ServletException{
    res.setContentType("text/html");
    PrintWriter out=res.getWriter(); out.print("<html><body>"); out.print("<b>hello
    generic servlet</b>"); out.print("</body></html>");
    }
}
```

HttpServlet class

1. HttpServlet class
2. Methods of HttpServlet class

The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

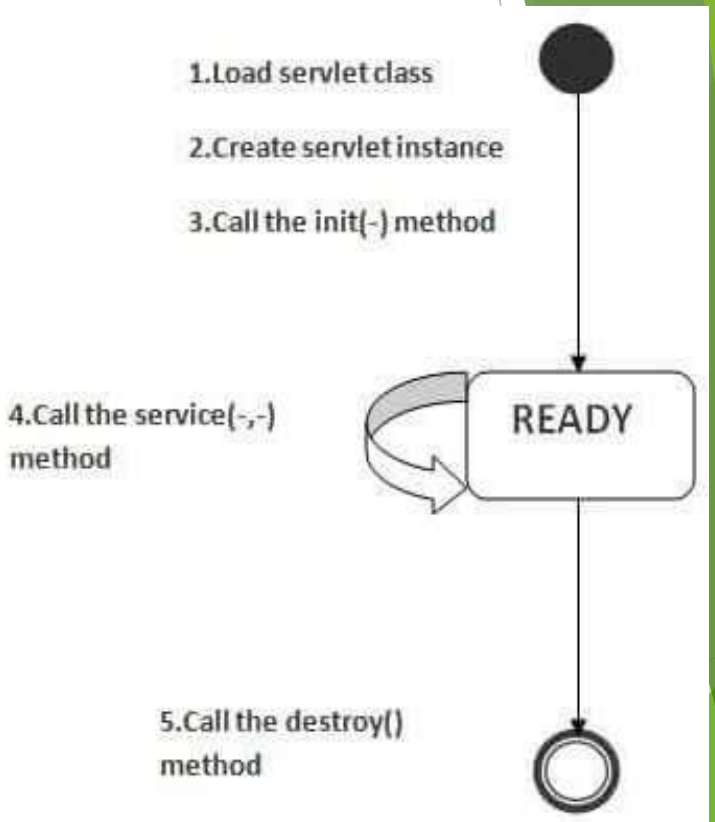
Methods of HttpServlet class

There are many methods in HttpServlet class.They are as follows:

- **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
- **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
- **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
- **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
- **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
- **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
- **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
- **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
- **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
- **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

Life Cycle of a Servlet (Servlet Life Cycle)

- Servlet class is loaded.
- Servlet instance is created.
- init method is invoked.
- service method is invoked.
- destroy method is invoked.



As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

1) Servlet class is loaded

The classloader is responsible to load the servlet class.The servlet class is loaded when the first request for the servlet is received by the web container.

2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class.The servlet instance is created only once in the servlet life cycle.

3) init method is invoked

The web container calls the init method only once after creating the servlet instance.The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

public void init(/ervletConfig config) throws /ervletException

4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once.The syntax of the service method of the Servlet interface is given below:

public void service(/ervletRequest request,/ervletResponse response) throws /ervletException, IOException

5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc.The syntax of the destroy method of the Servlet interface is given below:

public void destroy()

Steps to create a servlet example

There are given 6 steps to create a **servlet example**. These steps are required for all the servers.

- Create a directory structure
- Create a Servlet
- Compile the Servlet
- Create a deployment descriptor
- Start the server and deploy the application
- Accessing the servlet

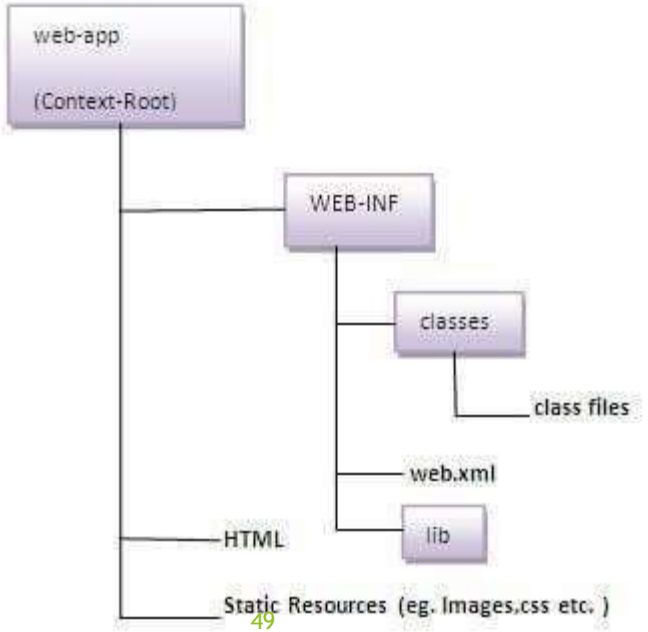
The servlet example can be created by three ways:

- By implementing Servlet interface,
- By inheriting GenericServlet class, (or)
- By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc. Here, we are going to use **apache tomcat server** in this example.

1) Create a directory structures

- The directory structure defines that where to put the different types of files so that web container may get the information and respond to the client.
- The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.
- As you can see that the servlet class file m
  folder.The web.xml file must be under the



2)Create a Servlet

There are three ways to create the servlet.

1. By implementing the Servlet interface
2. By inheriting the GenericServlet class
3. By inheriting the HttpServlet class

- The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.
- In this example we are going to create a servlet that extends the HttpServlet class. In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method. Notice that get request is the default request.

DemoServlet.java

```
import javax.servlet.http.*; import
javax.servlet.*; import java.io.*;
public class DemoServlet extends HttpServlet{
        public void doGet(HttpServletRequest req,HttpServletResponse res)
        throws ServletException,IOException{
        res.setContentType("text/html");//setting the content type
        PrintWriter pw=res.getWriter();//get the stream to write the data10.
        //writing html in the stream
        pw.println("<html><body>");
        pw.println("Welcome to servlet");
        pw.println("</body></html>");
        pw.close();//closing the stream
        }
}
```

3)Compile the servlet

- For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files:

Jar file	Server
1) servlet-api.jar	ApacheTomcat
2) weblogic.jar	Weblogic
3) javaee.jar	Glassfish
4) javaee.jar	JBoss

Two ways to load the jar file

1. set classpath
2. paste the jar file in JRE/lib/ext folder

- Put the java file in any folder. After compiling the java file, paste the class file of servlet in WEB-INF/classes directory.

4)Create the deployment descriptor (web.xml file)

- The deployment descriptor is an xml file, from which Web Container gets the information about the servet to be invoked.
- The web container uses the Parser to get the information from the web.xml file.There are many xml parsers such as SAX, DOM and Pull.
- There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

Web.xml

- **web.xml file**
- **<web-app>**

- **<servlet>**
- **<servlet-name>**sonoojaiswal**</servlet-name>**
- **<servlet-class>**DemoServlet**</servlet-class>**
- **</servlet>**

- **<servlet-mapping>**
- **<servlet-name>**sonoojaiswal**</servlet-name>**
- **<url-pattern>**/welcome**</url-pattern>**
- **</servlet-mapping>**

- **</web-app>**

There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file.The elements are as follows:

- <web-app> represents the whole application.
- <servlet> is sub element of <web-app> and represents the servlet.
- <servlet-name> is sub element of <servlet> represents the name of the servlet.
- <servlet-class> is sub element of <servlet> represents the class of the servlet.
- <servlet-mapping> is sub element of <web-app>. It is used to map the servlet.
- <url-pattern> is sub element of <servlet-mapping>.This pattern is used at client side to invoke the servlet.

5)Start the Server and deploy the project

- To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.
- One Time Configuration for Apache Tomcat Server

You need to perform 2 tasks:

1. set JAVA_HOME or JRE_HOME in environment variable (It is required to start server).
2. Change the port number of tomcat (optional). It is required if another server is running on same port (8080).

6) How to access the servlet

- Open broser and write http://hostname:portno/contextroot/urlpatternofservlet. For example:
    1. http://localhost:9999/demo/welcome

Working of Servlet

It is important to learn how servlet works for understanding the servlet well. Here, we are going to get the internal detail about the first servlet program.

The server checks if the servlet is requested **for the first time**.

If yes, web container does the following tasks:

loads the servlet class.

instantiates the servlet class.

calls the init method passing the ServletConfig object

else

calls the service method passing request and response objects

The web container calls the destroy method when it needs to remove the servlet such as at time of stopping server or undeploying the project.

How web container handles the servlet request?

The web container is responsible to handle the request. Let's see how it handles the request.

- maps the request with the servlet in the web.xml file.
- creates request and response objects for this request
- calls the service method on the thread
- The public service method internally calls the protected service method
- The protected service method calls the doGet method depending on the type of request.
- The doGet method generates the response and it is passed to the client.
- After sending the response, the web container deletes the request and response objects.The thread is contained in the thread pool or deleted depends on the server implementation.

What is written inside the public service method?

The public service method converts the ServletRequest object into the HttpServletRequest type and ServletResponse object into the HttpServletResponse type.Then, calls the service method passing these objects. Let's see the internal code:

```
public void service(ServletRequest req, ServletResponse res)
      throws ServletException, IOException
  {
     HttpServletRequest request;
     HttpServletResponse response;
     try
     {
        request = (HttpServletRequest)req;
        response = (HttpServletResponse)res;
     }
     catch(ClassCastException e)
     {
        throw new ServletException("non-HTTP request or response");
     }
     service(request, response);
  }
```

What is written inside the protected service method?

The protected service method checks the type of request, if request type is get, it calls doGet method, if request type is post, it calls doPost method, so on. Let's see the internal code:

```
protected void service(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException
{
    String method = req.getMethod();
    if(method.equals("GET"))
    {
        long lastModified = getLastModified(req);
        if(lastModified == -1L)
        {
            doGet(req, resp);
        }
....
//rest of the code
    }
}
```

War File

What is war file?

web archive (war) file contains all the contents of a web application. It reduces the time duration for transferring file.

Advantage of war file

saves time: The war file combines all the files into a single unit. So it takes less time while transferring file from client to server.

How to create war file?

To create war file, you need to use jar tool of JDK. You need to use -c switch of jar, to create the war file.

Go inside the project directory of your project (outside the WEB-INF), then write the following command:

1. jar -cvf projectname.war *

Here, -c is used to create file, -v to generate the verbose output and -f to specify the arhive file name.

The * (asterisk) symbol signifies that all the files of this directory (including sub directory).

How to deploy the war file?

There are two ways to deploy the war file.

1. By server console panel
2. By manually having the war file in specific folder of server.

If you want to deploy the war file in apache tomcat server manually, go to the webapps directory of apache tomcat and paste the war file here.

Now, you are able to access the web project through browser.

How to extract war file manually?

To extract the war file, you need to use -x switch of jar tool of JDK. Let's see the command to extract the war file.

1. jar -xvf projectname.war

welcome-file-list in web.xml

The welcome-file-list element of web-app, is used to define a list of welcome files. Its sub element is welcome-file that is used to define the welcome file.

A welcome file is the file that is invoked automatically by the server, if you don't specify any file name.

By default server looks for the welcome file in following order:

1. welcome-file-list in web.xml
2. index.html
3. index.htm
4. index.jsp

If none of these files are found, server renders 404 error.

If you have specified welcome-file in web.xml, and all the files index.html, index.htm and index.jsp exists, priority goes to welcome-file.

If welcome-file-list entry doesn't exist in web.xml file, priority goes to index.html file then index.htm and at last index.jsp file.

<web-app>

….

<welcome-file-list>

<welcome-file>home.html**</welcome-file>**

<welcome-file>default.html**</welcome-file>**

</welcome-file-list>

</web-app>

Now, home.html and default.html will be the welcome files.

If you have the welcome file, you can directory invoke the project as given below:

http://localhost:8888/myproject

Handling GET and POST Requests

To handle HTTP requests in a servlet, extend the HttpServlet class and override the servlet methods that handle the HTTP requests that your servlet supports. This lesson illustrates the handling of GET and POST requests. The methods that handle these requests are doGet and doPost.

Handling GET requests

- Handling GET requests involves overriding the doGet method. The following example shows the BookDetailServlet doing this. The methods discussed in the Requests and Responses section are shown in bold.

- Within the doGet method, the getParameter method gets the servlet's expected argument.
- To respond to the client, the example doGet method uses a Writer from the HttpServletResponse object to return text data to the client. Before accessing the writer, the example sets the content-type header. At the end of the doGet method, after the response has been sent, the Writer is closed.

```
public class BookDetailServlet extends HttpServlet {
   public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
      throws ServletException, IOException
   {       ...
// set content-type header before accessing the Writer
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
                // then write the response
      out.println("<html>" +
                "<head><title>Book Description</title></head>" +
                ...);
      //Get the identifier of the book to display
      String bookId = request.getParameter("bookId");
      if (bookId != null) {
         // and the information about the book and print it
         ...       }
      out.println("</body></html>");
      out.close();
          }...   }
```

The servlet extends the HttpServlet class and overrides the doGet method.

Handling POST Requests

- Handling POST requests involves overriding the doPost method. The following example shows the ReceiptServlet doing this. Again, the methods discussed in the Requests and Responses section are shown in bold.
- The servlet extends the HttpServlet class and overrides the doPost method. Within the doPost method, the getParameter method gets the servlet's expected argument.
- To respond to the client, the example doPost method uses a Writer from the HttpServletResponse object to return text data to the client. Before accessing the writer the example sets the content-type header. At the end of the doPost method, after the response has been set, the Writer is closed

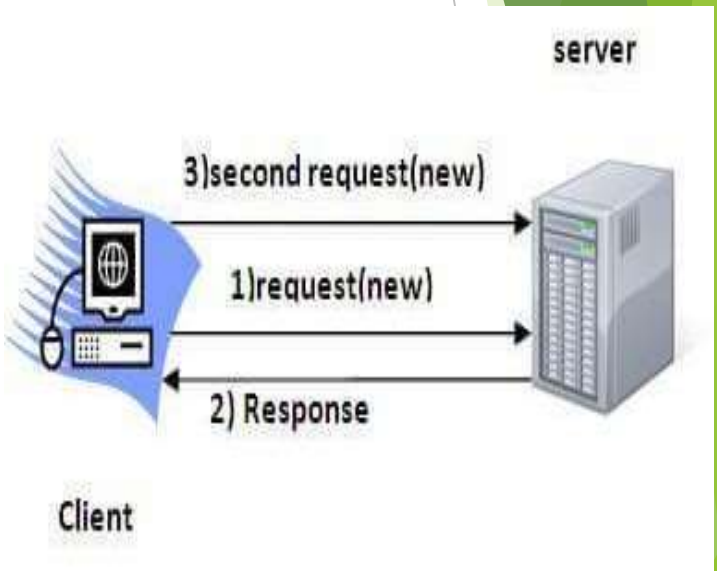
```
public class ReceiptServlet extends HttpServlet {
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
              throws ServletException, IOException  {       …
    / /  set content type header before accessing the Writer
    response.setContentType("text/html");
    PrintWriter out =  response.getWriter();
    / /  then write the response
    out.println("<html>" +
                "<head><title> Receipt </title>" +
                …);
    out.println("<h3>Thank you for purchasing your books from us " +
                request.getParameter("cardname") +
                …);
   out.close();
}    …   }
```

Session Tracking in Servlets

1. Session Tracking
2. Session Tracking Techniques

Session simply means a particular interval of time.

- Session Tracking is a way to maintain state (data) of an user. It is also known as session management in servlet.
- Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.
- HTTP is stateless that means each request is considered as the new request. It is shown in the given figure:



Why use Session Tracking?

To recognize the user It is used to recognize the particular user.

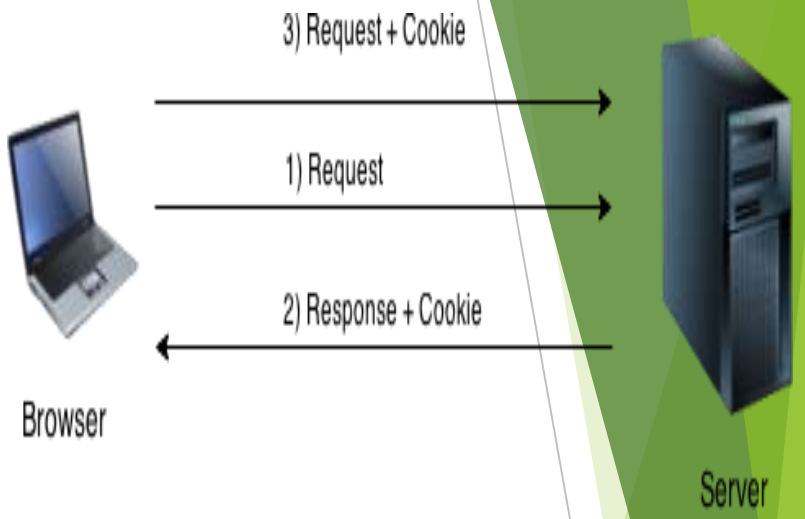
Session Tracking Techniques

There are four techniques used in Session tracking:

1. Cookies
2. Hidden Form Field
3. URL Rewriting
4. HttpSession

Cookies in Servlet

- A **cookie** is a small piece of information that is persisted between the multiple client requests.
- A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.



How Cookie works

- By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.

Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

It is valid for single session only. It is removed each time when user closes the browser.

Persistent cookie

It is valid for multiple session . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

Cookie class

- javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Useful Methods of Cookie class

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie.The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces.They are:

1. public void addCookie(Cookie ck):method of HttpServletResponse interface is used to add cookie in response object.
2. public Cookie[] getCookies():method of HttpServletRequest interface is used to return all the cookies from the browser.

How to create Cookie?

Let's see the simple code to create cookie.

```
1. Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object
2. response.addCookie(ck);//adding cookie in the response
```

How to delete Cookie?

Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.

```
1.  Cookie ck=new Cookie("user","");//deleting value of cookie
2.  ck.setMaxAge(0);//changing the maximum age to 0 seconds
```

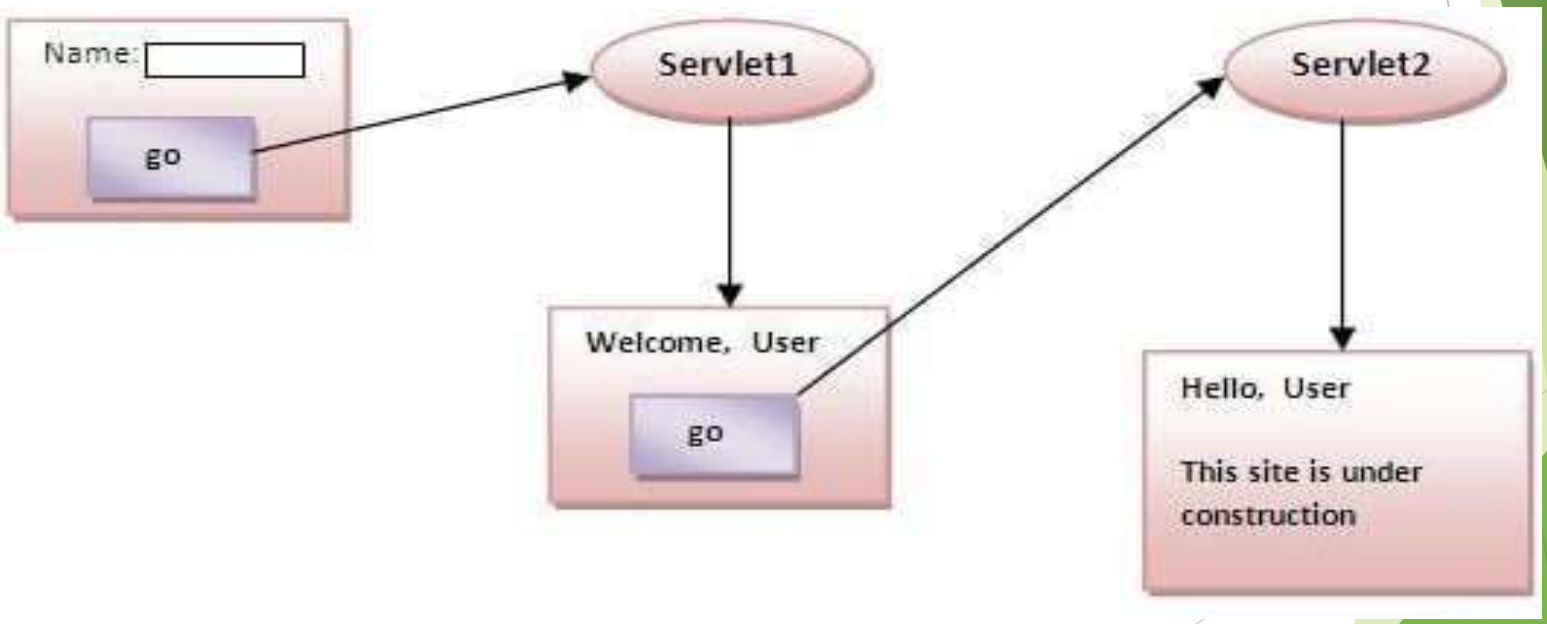
How to get Cookies?

Let's see the simple code to get all the cookies.

```
1.  Cookie ck[]=request.getCookies();
2.  for(int i=0;i<ck.length;i++){
3.  out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie
4.  } .  response.addCookie(ck);//adding cookie in the response
```

Simple example of Servlet Cookies

- In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



```
1.  <form action="servlet1" method="post">
2.  Name:<input type="text" name="userName"/><br/>
3.  <input type="submit" value="go"/>
4.  </form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
  public void doPost(HttpServletRequest request, HttpServletResponse response){
try{
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   String n=request.getParameter("userName");
   out.print("Welcome "+n);
   Cookie ck=new Cookie("uname",n);//creating cookie object
   response.addCookie(ck);//adding cookie in the response
   //creating submit button
   out.print("<form action='servlet2'>");
   out.print("<input type='submit' value='go'>");
   out.print("</form>");
   out.close();
      }catch(Exception e){System.out.println(e);}
  }
}
```

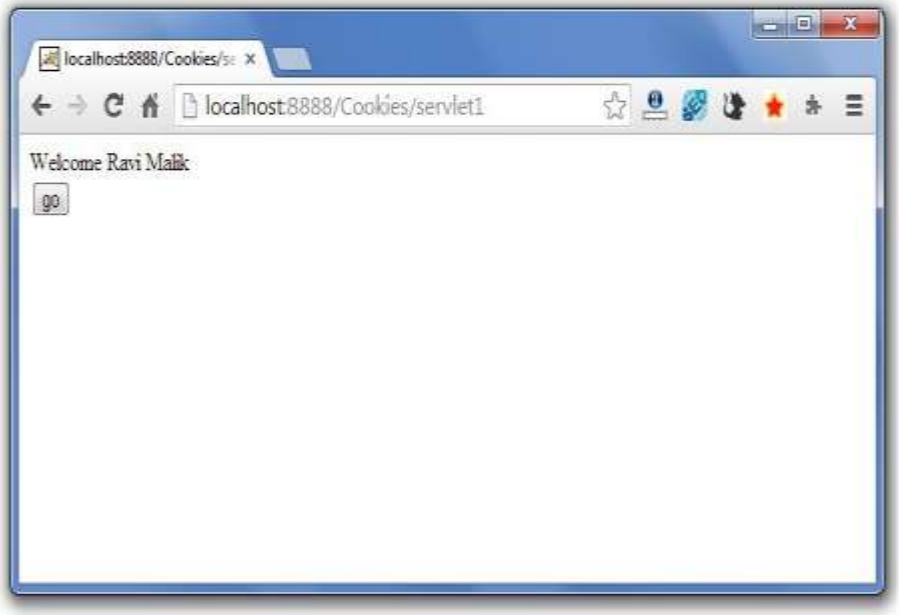
SecondServlet.java

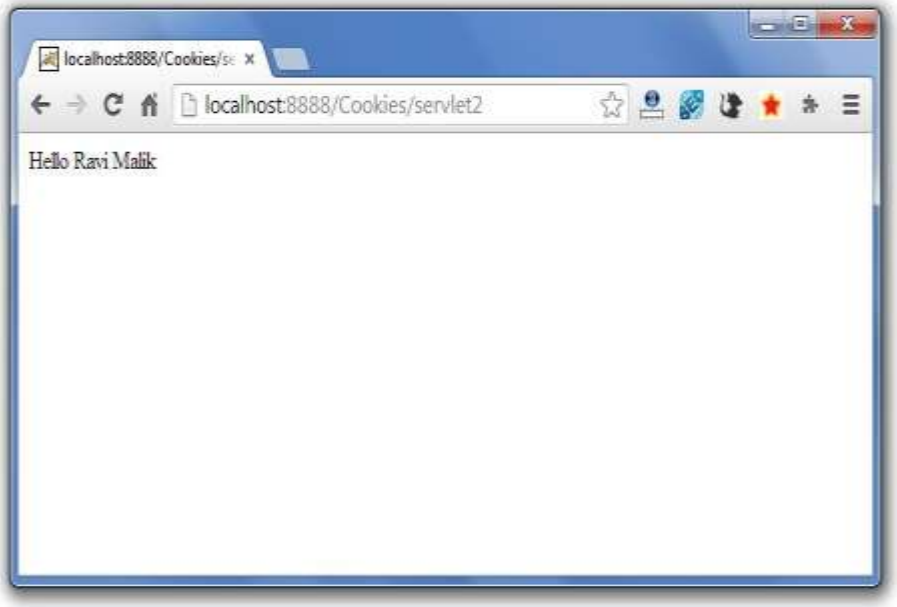
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse
response){
    try{

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    Cookie ck[]=request.getCookies();
    out.print("Hello "+ck[0].getValue());

    out.close();

        }catch(Exception e){System.out.println(e);}
    }
```

web.xml

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```


localhost:8888/Cookies/
localhost:8888/Cookies/
Name: Ravi Malik
go


localhost:8888/Cookies/se
localhost:8888/Cookies/servlet1
Welcome Ravi Malik
go


localhost:8888/Cookies/se
localhost:8888/Cookies/servlet2
Hello Ravi Malik