7 MVC Architecture

CONTENTS

Model-View-Controller (MVC) is a widely used software design pattern that was created by Xerox PARC for Smalltalk-80 in the 1980s. More recently, it has become the recommended model for Sun's J2EE platform, and it is gaining increasing popularity among ColdFusion and PHP developers. The MVC pattern is a useful addition to a toolkit, no matter what language you choose.

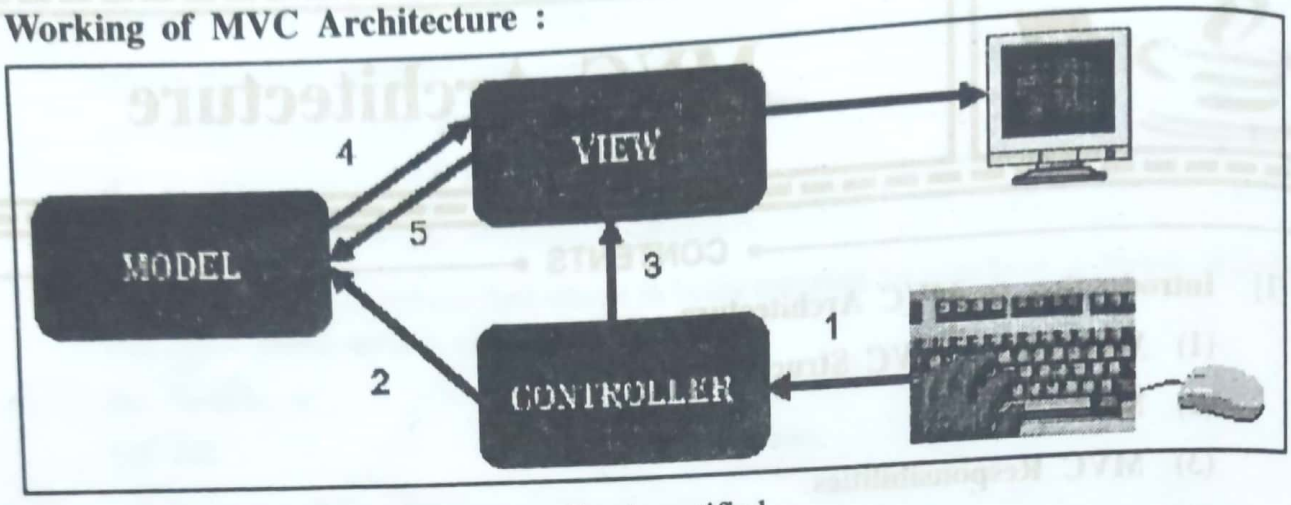
MVC - Model-View-Controller - is a design pattern for the architecture of web applications. It is a widely adopted pattern, across many languages and implementation frameworks, whose purpose is to achieve a clean separation between three components of most any web application.

[1] INTRODUCTION OF MVC ARCHITECTURE

MVC stands for Model – View – Controller :

1. The **Model** represents the structure of the data in the application, as well as application-specific operations on those data. **-- Notify view about data updates.**
2. The **View** renders the contents of a model. It specifies exactly how the model data should be presented. **-- Change rendering as needed.**
3. The **Controller** translates user actions (mouse motions, keystrokes, words spoken, etc.) and user input into application function calls on the model, and selects the appropriate View based on user preferences and Model state. **-- Select view to be rendered based on event notifications and method invocations.**

(1) Working of MVC Architecture :



1. When user performs action, controller is notified.
2. Controller may request changes to model.
3. Controller may tell view to update.
4. Model may notify view if it has been modified.
5. View may need to query model for current data.
6. View updates display for user.

(2) MVC Pattern :

1. Model :

1. Contains application & its data.
2. Provide methods to access & update data.
3. Interface defines allowed interactions.
4. Fixed interface enable both model & GUIs to be easily pulled out and replaced.
5. Examples :
   - Text documents
   - Spreadsheets
   - Web browser
   - Video games

2. View :

1. Provides visual representation of model.
2. Multiple views can display model at same time.
3. When model is updated, all its views are informed & given chance to update themselves.

3. Controller :

1. Users interact with the controller.
2. Interprets mouse movement, keystrokes, etc.
3. Communicates those activities to the model.
4. Interaction with model indirectly causes view(s) to update.

(3) MVC Responsibilities :

1. Model Responsibilities :

1. Store data in properties.
2. Implement application methods.
3. Provide methods to register/unregister views.
4. Notify views of state changes.

2. View Responsibilities :

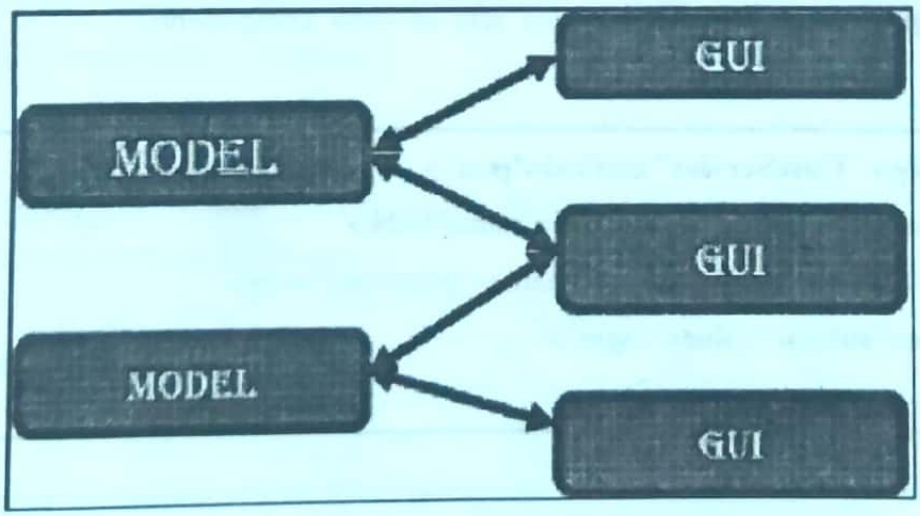
1. Create interface.
2. Update interface when model changes.
3. Forward input to controller.

3. Controller Responsibilities :

1. Translate user input into changes in the model.
2. If change is purely cosmetic, update view.

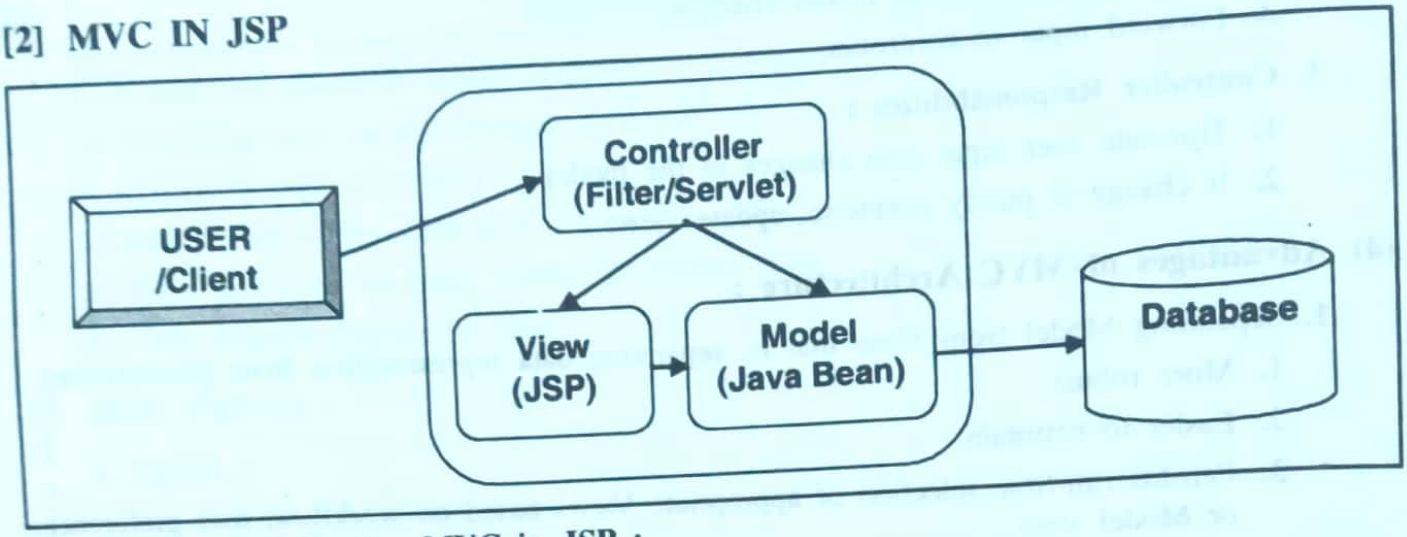
(4) Advantages of MVC Architecture :

1. Separating Model from View that is, separating data representation from presentation.
   1. More robust.
   2. Easier to maintain.
   3. Permits run-time selection of appropriate Views based on workflow, user preferences, or Model state.
2. Separating Controller from Model, application behavior from data representation.
3. Easy to add multiple data presentations for the same data.
   1. Multi-view applications.
   2. Different users.
   3. Different UI platforms (mobile, client-side, server-side,...).
   4. Alternate designs.



4. Model and View components can vary independently enhancing maintainability, extensibility, and testability.
5. Allows **user interfaces or views** to be easily added, removed, or changed.
6. Allows response to user input or **controller** to be easily changed.
7. Changes can happen dynamically at runtime.
8. Allows **multiple developers** to simultaneously update the interface, logic, or input of an application without affecting other source code.

[2] MVC IN JSP



Example of following MVC in JSP :

In this example, we are using servlet as a controller, jsp as a view component, Java Bean class as a model.

In this example, we have created 5 pages :

1. **index.jsp** a page that gets input from the user.
2. **CoreServlet.java** a servlet that acts as a controller.
3. **LoginBean.java** a bean file.
4. **Login.jsp** and **Error_login.jsp** files acts as view components.

Index.jsp :

2.

```
<form action="CoreServlet" method="post">
User Name:<input type="text" name="name"><br>
Password :<input type="password" name="password"><br>
<input type="submit" value="login">
</form>
```

CoreServlet.java :

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.http.*;
public class ControllerServlet extends HttpServlet
{
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
{
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        String name=request.getParameter("name");
        String password=request.getParameter("password");
        LoginBean bean=new LoginBean( );
        bean.setName(name);
        bean.setPassword(password);
        request.setAttribute("bean",bean);
        boolean status=bean.validate( );
        if(status)
        {
            RequestDispatcher rd=request.getRequestDispatcher("Login.jsp");
            rd.forward(request, response);
        }
       else
        {
            RequestDispatcher rd=request.getRequestDispatcher("Error_login.jsp");
            rd.forward(request, response);
        }
   }
   protected void doGet(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException
   {
       doPost(req, resp);
   }
}
```

LoginBean.java :

```
public class LoginBean
{
        private String name,password;
        public String getName( ) {  return name;  }
        public void setName(String name) {  this.name = name;  }
        public String getPassword( ) {  return password;  }
        public void setPassword(String password) {  this.password = password;  }
        public boolean validate( )
        {
            if(password.equals("admin"))
            {
                return true;
            }
            else
            {
                return false;
            }
        }
}
```

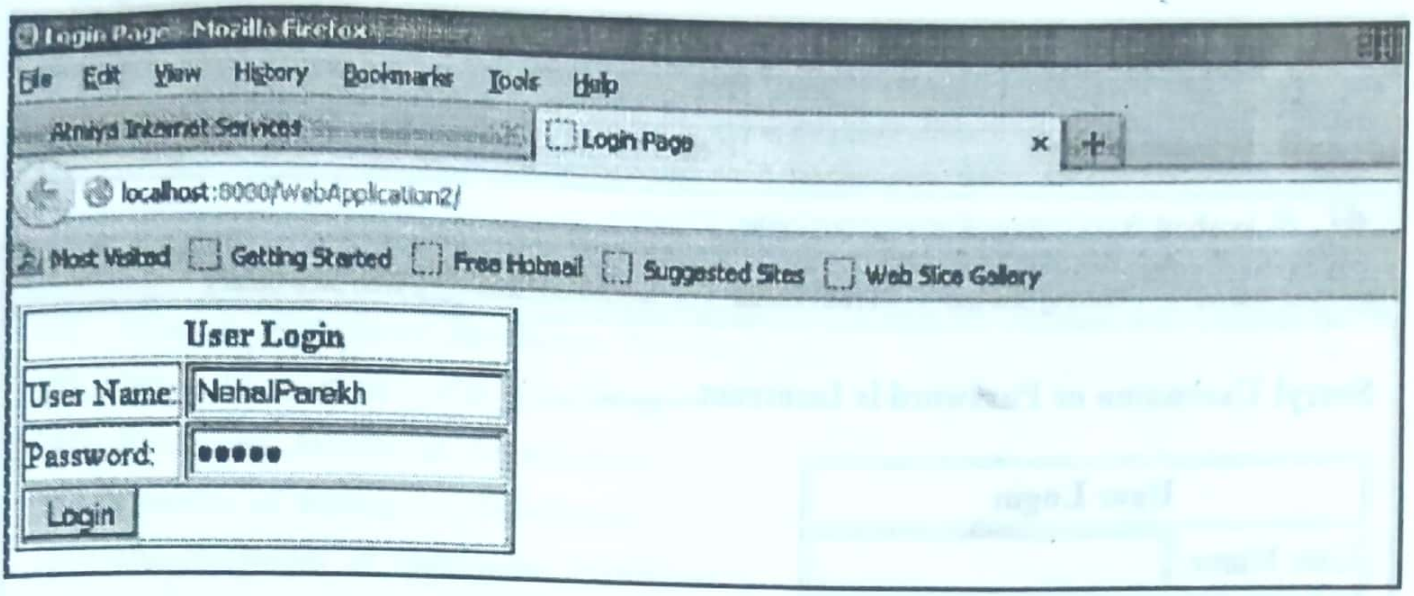
Login.jsp :

```
<%@page import="LoginBean"%>
<p> You are successfully logged in!!!!</p>
<%
        LoginBean bean = (LoginBean) request.getAttribute ("bean");
        out.print ("Welcome, "+bean.getName( ));
%>
```

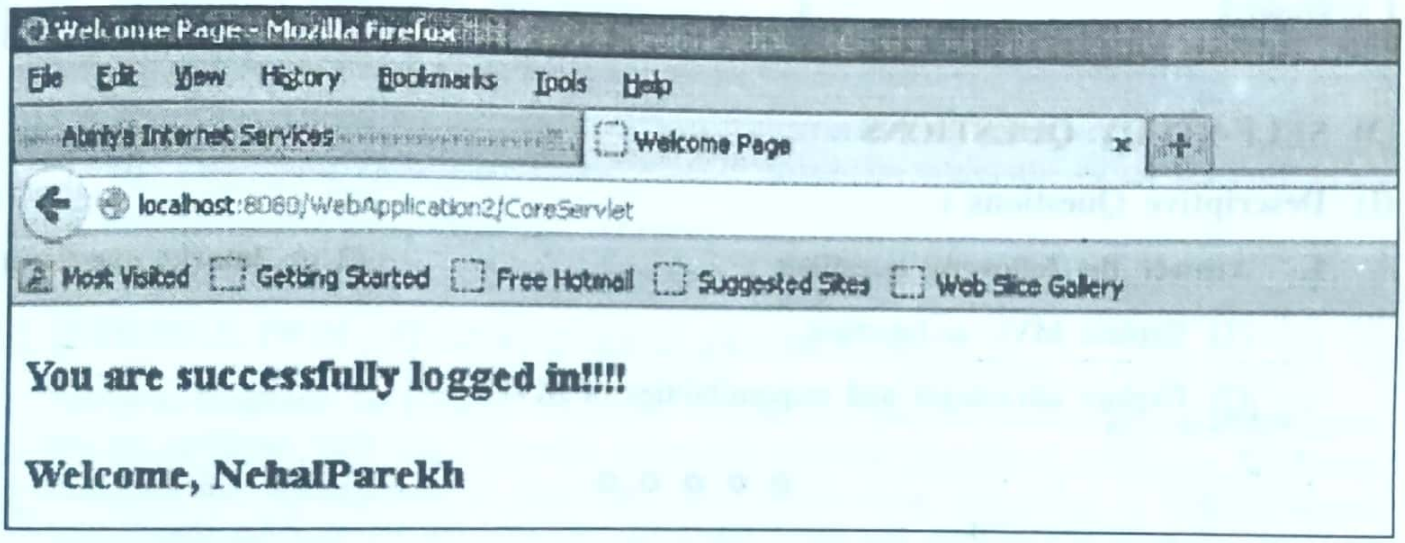
Error_login.jsp :

```
<p> Sorry! Username or Password is Incorrect……..</p>
<%@ include file="index.jsp" %>
```

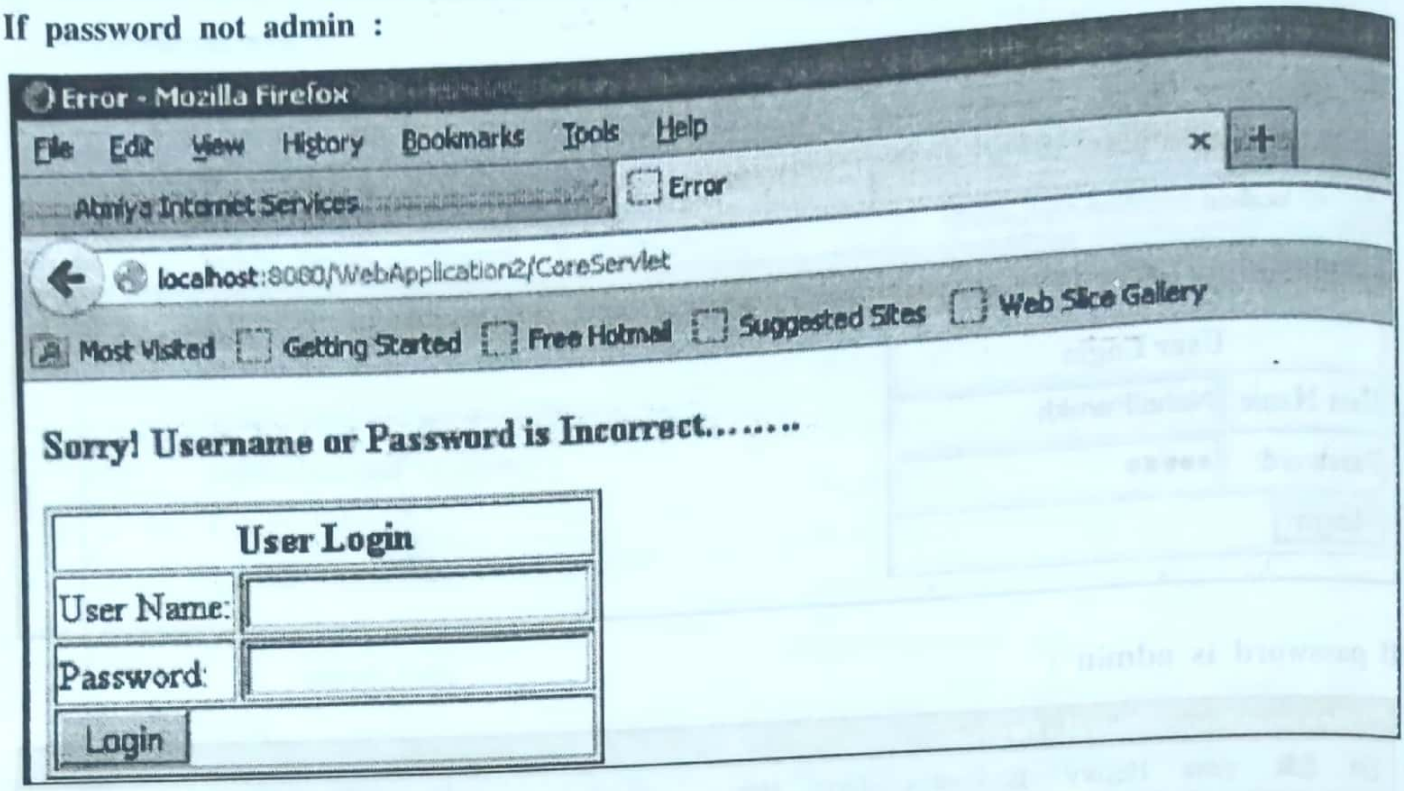
Output :



If password is admin :



If password not admin :



[3] SELF-STUDY QUESTIONS

(I) Descriptive Questions :

1. Answer the following questions : **(2 or 3marks questions)**

(1) Explain MVC architecture.

(2) Explain advantages and responsibilities of MVC.

✡ ✡ ✡ ✡ ✡

8 Enterprise JavaBeans

CONTENTS

[1] INTRODUCTION TO ENTERPRISE JAVABEANS

Enterprise JavaBeans are software component models; their purpose is to build/support enterprise specific problems. EJB - is a reusable server-side software component. Enterprise JavaBeans facilitates the development of distributed Java applications, providing an object-oriented transactional environment for building distributed, multi-tier enterprise components. An EJB is a remote object, which needs the services of an EJB container in order to execute.

EJB is a specification provided by Sun Microsystems to develop secured, robust and scalable distributed applications. The primary goal of an EJB is **WORA** (Write Once Run Anywhere). Enterprise Java Beans takes a high-level approach to building distributed systems. It frees the application developer and enables him/her to concentrate on programming only the business logic. For example, In an Employee management application method **calculateSal** is there by invoking this method remote client can access management services.

To run EJB application, you need an application server (EJB Container) such as Jboss, Glassfish, Weblogic, Websphere etc. EJB container performs following task :

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 and Web Services.

[2] WHAT DO ENTERPRISE JAVABEANS PROVIDE ?

In J2EE all the components run inside their own containers. JSP, Servlets and JavaBeans have their own web container. Similarly EJBs run inside EJB container. The container provides certain built-in services to EJBs which the EJBs use to function. The services that EJB container provides are :

1. Component Pooling
2. Resource Management
3. Transaction Management
4. Security
5. Persistence
6. Handling of multiple clients

1. Component Pooling :

The EJB container handles the pooling of EJB components. If there are no requests for a particular EJB then the container will probably contain zero or one instance of that component in memory. If need arises then it will increase component instances to satisfy all incoming requests. Then again if number of requests decrease, container will decrease the component instances in the pool. The best thing is that the client is absolutely unaware of this component pooling and the container handles all this for you.

2. Resource Management :

The container is also responsible for maintaining database connection pools. It provides you a standard way of obtaining and returning database connections. The container also manages EJB environment references and references to other EJBs. The container manages following types of resources and makes them available to EJBs :

1. JDBC 2.0 Data Sources
2. JavaMail Sessions
3. JMS Queues and Topics
4. URL Resources
5. Legacy Enterprise Systems via J2EE Connector Architecture

3. Transaction Management :

This is probably the single most important factor of all. A transaction is a single unit of work, composed of one or more steps. If all the steps succeed then the transaction is committed otherwise it is rolled back. There are different types of transactions and it is absolutely unimaginable to not to use transactions in today's business environments. We will learn more about transactions in a separate article.

4. Security :

The EJB container provides its own authentication and authorization control, allowing only specific clients to interact with the business process. There is no need for you to create security architecture of your own, you are provided with a built-in system, and all you have to do is to use it.

5. Persistence :

The container if desired can also maintain persistent data of our application. The container is then responsible for retrieving and saving the data for us while taking care of concurrent access from multiple clients and not corrupting the data.

6. Handling of Multiple Clients :

The EJB container handles multiple clients of different types. A JSP based thin client can interact with EJBs with same ease as that of GUI based thick client. The container is smart enough to allow even non-Java clients like COM based applications to interact with the EJB system. Like before the EJB container handles it all for you.

[3] WHEN TO USE ENTERPRISE JAVABEANS ?

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

Difference between RMI and EJB :

Both RMI and EJB, provides services to access an object running in another JVM (known as remote object) from another JVM. The differences between RMI and EJB are given below :

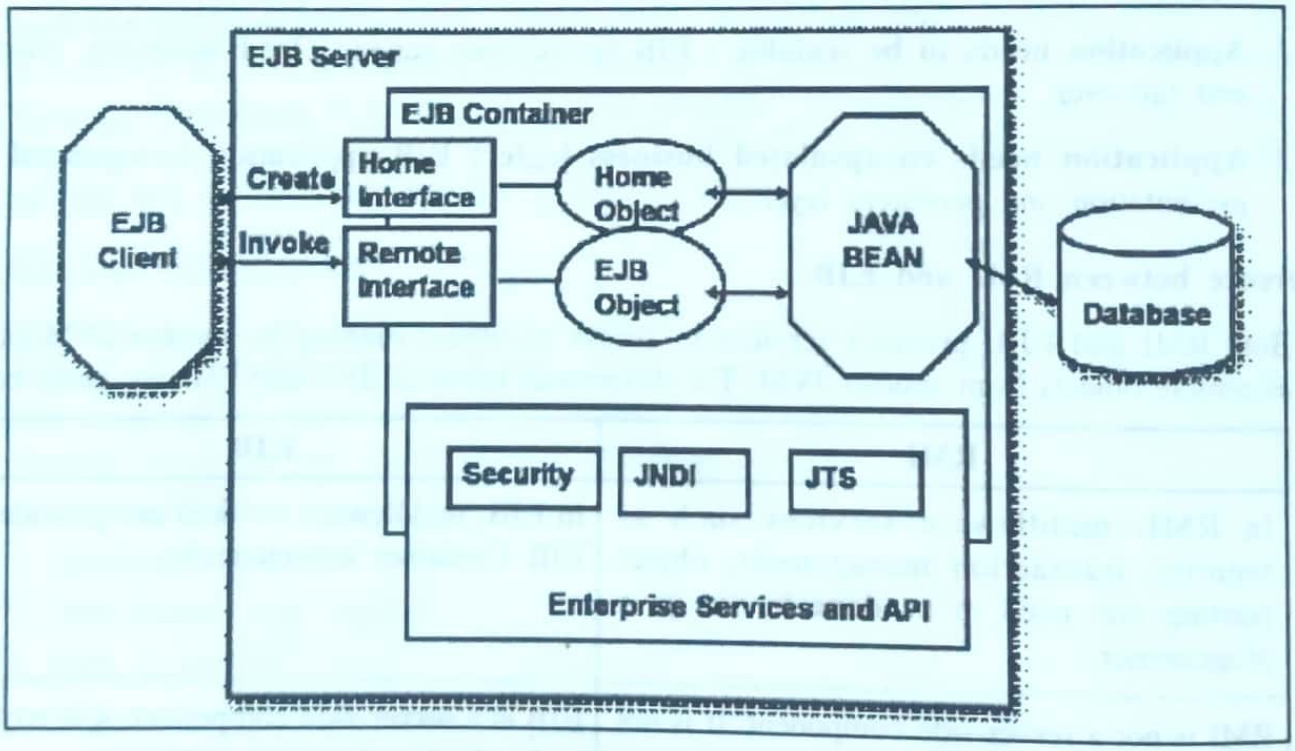
RMI	EJB
In RMI, middleware services such as security, transaction management, object pooling etc. need to be done by the java programmer.	In EJB, middleware services are provided by EJB Container automatically.
RMI is not a server-side component. It is not required to be deployed on the server.	EJB is a server-side component, it is required to be deployed on the server.
RMI is built on the top of socket programming.	EJB technology is built on the top of RMI.

EJB vs. JavaBeans :

JavaBeans	EJB
In JavaBeans the runtime execution environment provides services like Java libraries, Java application etc.	The EJB runtime environment provides services of Persistence, declarative transactions and security, connection pooling and lifecycle services.
The JavaBeans architecture is meant to provide a format for general-purpose components.	The EJB architecture provides a format for encapsulation and management of business logic.
JavaBeans has tier of execution at Client.	EJB has at Server (specifically business logic tier)

[4] ENTERPRISE JAVABEANS ARCHITECTURE

The Enterprise JavaBeans spec defines a server component model and specifies how to create server-side, scalable, transactional, and multiuser and secure enterprise-level components. Most important, EJBs can be deployed on top of existing transaction processing systems including traditional transaction processing monitors, Web, database and application servers.



1. **EJB clients :** EJB client applications utilize the Java Naming and Directory Interface (JNDI) to look up references to home interfaces and use home and remote EJB interfaces to utilize all EJB-based functionality.

2. **EJB home interfaces (and stubs) :** EJB home interfaces provide operations for clients to create, remove, and find handles to EJB remote interface objects. Underlying stubs marshal home interface requests and unmarshal home interface responses for the client.

3. **EJB remote interfaces (and stubs) :** EJB remote interfaces provide business-specific client interface methods defined for a particular EJB. Underlying stubs marshal remote interface requests and unmarshal remote interface responses for the client.

4. **EJB implementations :** EJB implementations are the actual EJB application components implemented by developers to provide any application-specific business method invocation, creation, removal, finding, activation, passivation, database storage, and database loading logic.

5. **Container EJB implementations (skeletons and delegates) :** The container manages the distributed communication skeletons used to marshal and unmarshal data sent to and from the client. Containers may also store EJB implementation instances in a pool and use delegates to perform any service-management operations related to a particular EJB before calls are delegated to the EJB implementation instance.

[5] BENEFITS OF ENTERPRISE JAVABEANS

1. Enterprise beans simplify the development of large, distributed applications.

2. **Because** First the EJB container provides system-level services to enterprise beans, the bean developer can concentrate on solving business problems.

3. The EJB container, rather than the bean developer, is responsible for system-level services such as transaction management and security authorization.

4. **Second**, because the beans rather than the clients contain the application's business logic, the client developer can focus on the presentation of the client. The client developer does not have to code the routines that implement business rules or access databases. As a result, the clients are thinner, a benefit that is particularly important for clients that run on small devices.

5. **Third**, because enterprise beans are portable components, the application assembler can build new applications from existing beans.

6. Other advantages are :
   (1) Simplicity
   (2) Application portability
   (3) Component reusability
   (4) Ability to build complex applications
   (5) Separation of business logic from presentation logic
   (6) Deployment in many operating environments
   (7) Distributed deployment
   (8) Application interoperability
   (9) Integration with non-Java systems
   (10) Educational resources and development tools

[6] DISADVANTAGES OF ENTERPRISE JAVABEANS

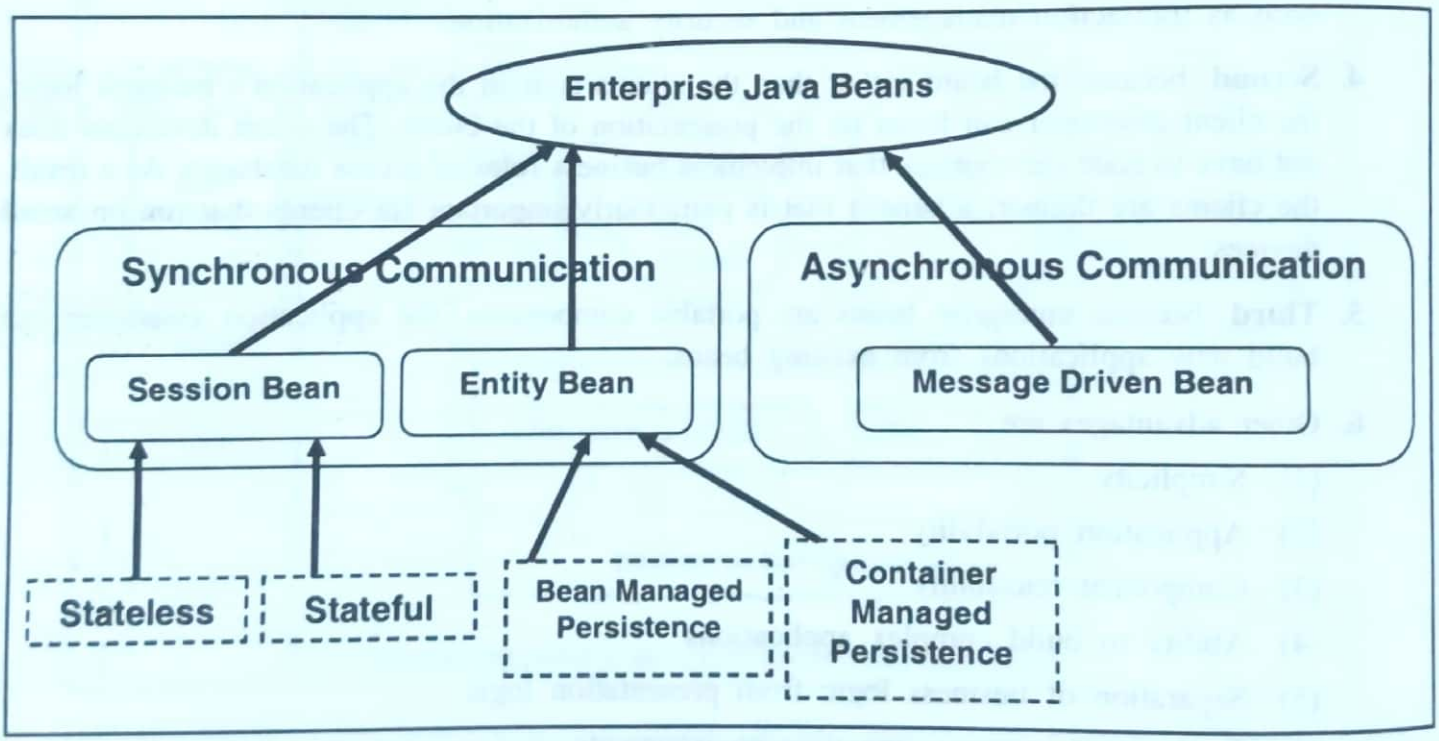
1. Requires application server.
2. Requires only java client. For other language client, you need to go for web service.
3. Complex to understand and develop EJB applications.

[7] TYPES OF ENTERPRISE BEAN

EJBs are distinguished along three main functional roles. Within each primary role, the EJBs are further distinguished according to sub roles. By partitioning EJBs into roles, the programmer can develop an EJB according to a more focused programming model than, if, for instances such roles were not distinguished earlier. These roles also allow the EJB container to determine the best management of a particular EJB based on its programming model type.

There are three main types of beans :

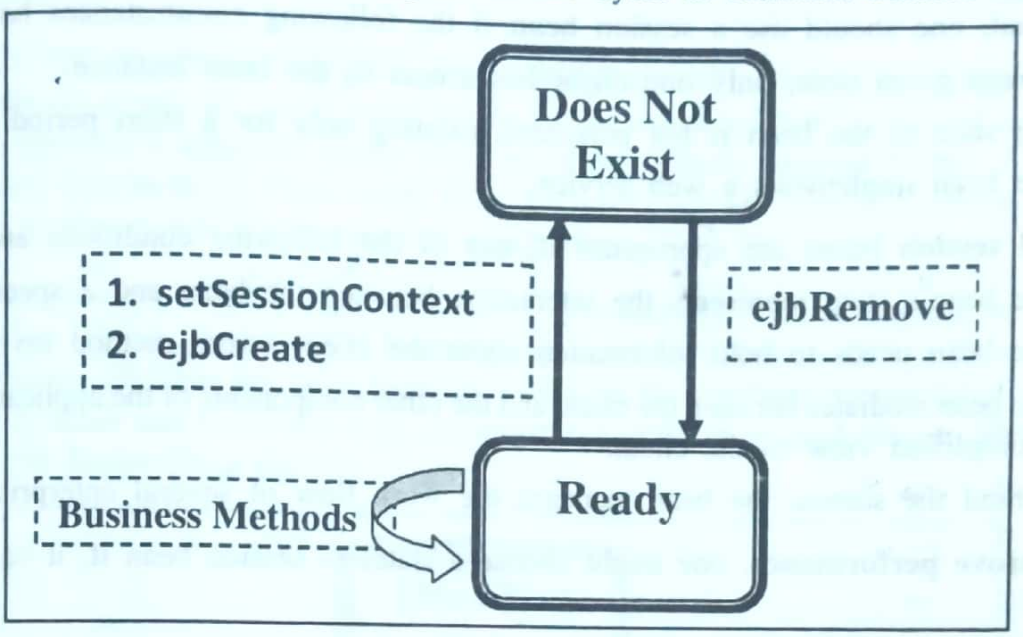
1. Session Bean
   (1) Stateless Session Bean
   (2) Stateful Session
2. Entity Beans
3. Message-driven Beans



1. Session Bean :

A session EJB is a non persistent object. Its lifetime is the duration of a particular interaction between the client and the EJB. The client normally creates an EJB, calls methods on it, and then removes it. If, the client fails to remove it, the EJB container will remove it after a certain period of inactivity. There are two types of session beans :

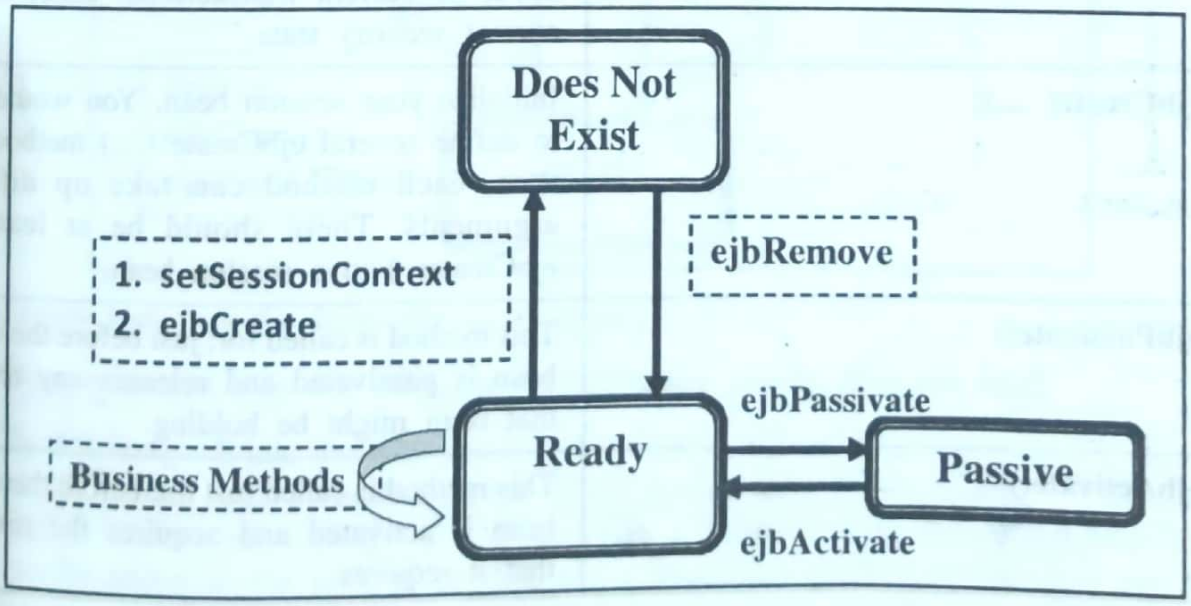
(1) **Stateless Session Beans :** A stateless session EJB is shared between a numbers of clients. It does not maintain conversational state. After each method call, the container may choose to destroy a stateless session bean, or recreate it, clearing itself out, of all the information pertaining the invocation of the last method. The algorithm for creating new instance or instance reuse is container specific. Following is the life cycle of Stateless Session Beans :



Does not exist : In this state, the bean instance simply does not exist.

Ready state : When EJB Server is first started, several bean instances are created and placed in the Ready pool. More instances might be created by the container as and when needed by the EJB container.

(2) **Stateful Session Beans :** A stateful session bean is a bean that is designed to service business processes that span multiple method requests or transaction. To do this, the stateful bean retains the state for an individual client. If, the stateful bean's state is changed during method invocation, then, that same state will be available to the same client upon invocation. Following is the life cycle of Stateful Session Beans :



Does not exist : In this state, the bean instance simply does not exist.

Ready state : Instance is tied to a particular client and engaged in a conversation.

Passive state : A bean instance in the passive state is passivated to conserve resources.

In general, one should use a **session bean** if the following circumstances hold :

(1) At any given time, only one client has access to the bean instance.

(2) The state of the bean is not persistent, existing only for a short period and therefore.

(3) The bean implements a web service.

Stateful session beans are appropriate if, any of the following conditions are true :

(1) The bean's state represents the interaction between the bean and a specific client.

(2) The bean needs to hold information about the client across method invocations.

(3) The bean mediates between the client and the other components of the application, presenting a simplified view to the client.

(4) Behind the scenes, the bean manages the work flow of several enterprise beans.

To **improve performance**, one might choose a stateless session bean if, it has any of these traits :

(1) The bean's state has no data for a specific client.

(2) In a single method invocation, the bean performs a generic task for all clients. For example, you might use a stateless session bean to send a promotional email to several registered users.

Required Methods in Session Bean :

The following are the required methods in a Session Bean :

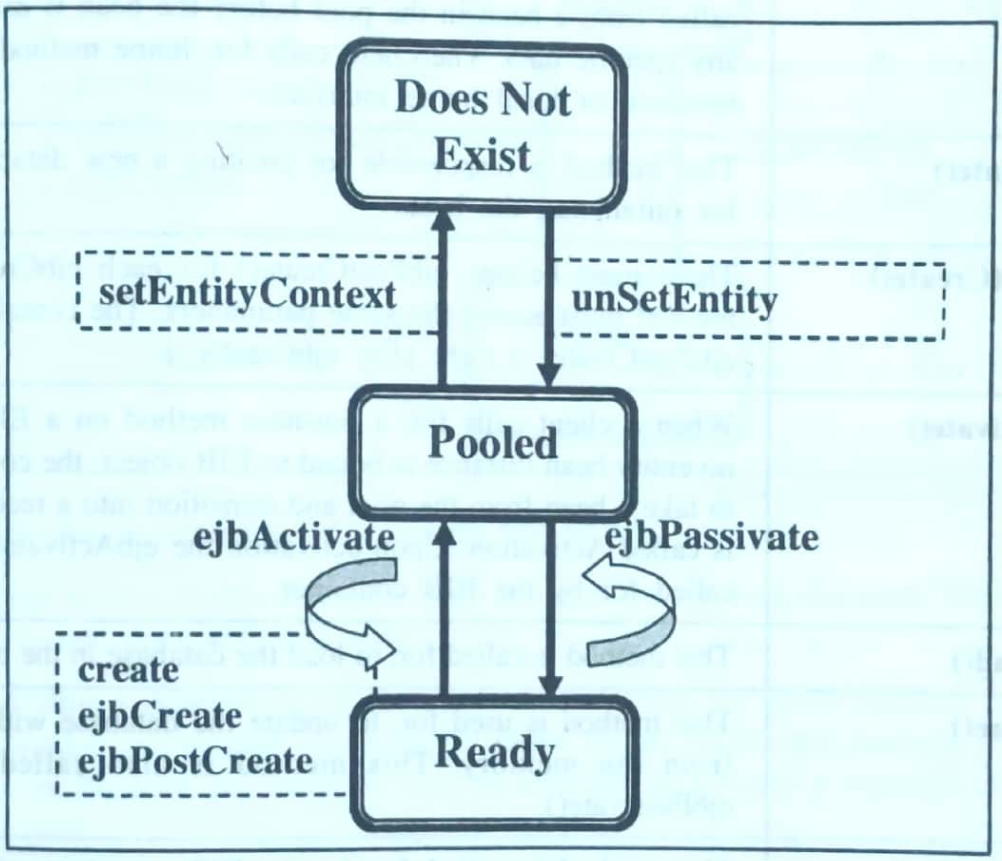
setSessionContext(SessionContext ctx)	Associate your bean with a session context. Your bean can make a query to the context about its current transactional state, and its current security state.
ejbCreate(...)	Initialize your session bean. You would need to define several ejbCreate (...) methods and then, each method can take up different arguments. There should be at least one ejbCreate( ) in a session bean.
ejbPassivate()	This method is called for; just before the session bean is passivated and releases any resource that bean might be holding.
ejbActivate()	This method is called just for, before the session bean is activated and acquires the resources that it requires.

ejbRemove()	This method is called for, by the ejb container just before the session bean is removed from the memory.

2. Entity Bean :

Entity EJBs represent persistent objects. Their lifetimes are not related to the duration of interaction with clients. In nearly all cases, entity beans are synchronized with relational databases. This is how persistence is achieved. Entity EJBs are always shared amongst clients. A client cannot get an entity EJB to itself. Thus, entity EJBs are nearly always used as a scheme for mapping relational databases into object-oriented applications.

An important feature of entity EJBs is that they have identity—that is, one can be distinguished from another. This is implemented by assigning a **primary key** to each instance of the EJB, where 'primary key' has the same meaning as it does for database management. Primary keys that identify EJBs can be of any type, including programmer-defined classes. Following is the life cycle of Entity Bean (MDB) :



An entity bean has the following three states :

1. **Does not exist :** In this state, the bean instance simply does not exist.
2. **Pooled state :** When the EJB server is first started, several bean instances are created and placed in the pool. A bean instance in the pooled state is not tied to a particular data, that is, it does not correspond to a record in a database table. Additional bean instances can be added to the pool as needed, and a maximum number of instances can be set.

3. **Ready state :** A bean instance in the ready state is tied to a particular data, that is, it represents an instance of an actual business object.

Required Methods in Entity Bean :

Entity beans can be bean managed or container managed. Here, are the methods that are required for entity beans :

setEntityContext( )	This method is called for, if a container wants to increase its pool size of bean instances, then, it will instantiate a new entity bean instance. This method associates a bean with context information. Once this method is called for, then, the bean can access the information about its environment.
ejbFind(..)	This method is also known as the Finder method. The Finder method locates one or more existing entity bean data instances in underlying persistent store.
ejbHome(..)	The Home methods are special business methods because they are called from a bean in the pool before the bean is associated with any specific data. The client calls for, home methods from home interface or local home interface.
ejbCreate()	This method is responsible for creating a new database data and for initializing the bean.
ejbPostCreate()	There must be one ejbPostCreate() for each ejbCreate( ). Each method must accept the same parameters. The container calls for, ejbPostCreate( ) right after ejbCreate( ).
ejbActivate()	When a client calls for, a business method on a EJB object but no entity bean instance is bound to EJB object, the container needs to take a bean from the pool and transition into a ready state. This is called Activation. Upon activation the ejbActivate( ) method is called for by the EJB container.
ejbLoad()	This method is called for, to load the database in the bean instance.
ejbStore()	This method is used for, to update the database with new values from the memory. This method is also called for during ejbPassivate().
ejbPassivate()	This method is called for, by the EJB container when an entity bean is moved from the ready state to the pool state.
ejbRemove()	This method is used to destroy the database data. It does not remove the object. The object is moved to the pool state for reuse.
unsetEntityContext()	This method removes the bean from its environment. This is called for, just before destroying the entity bean.

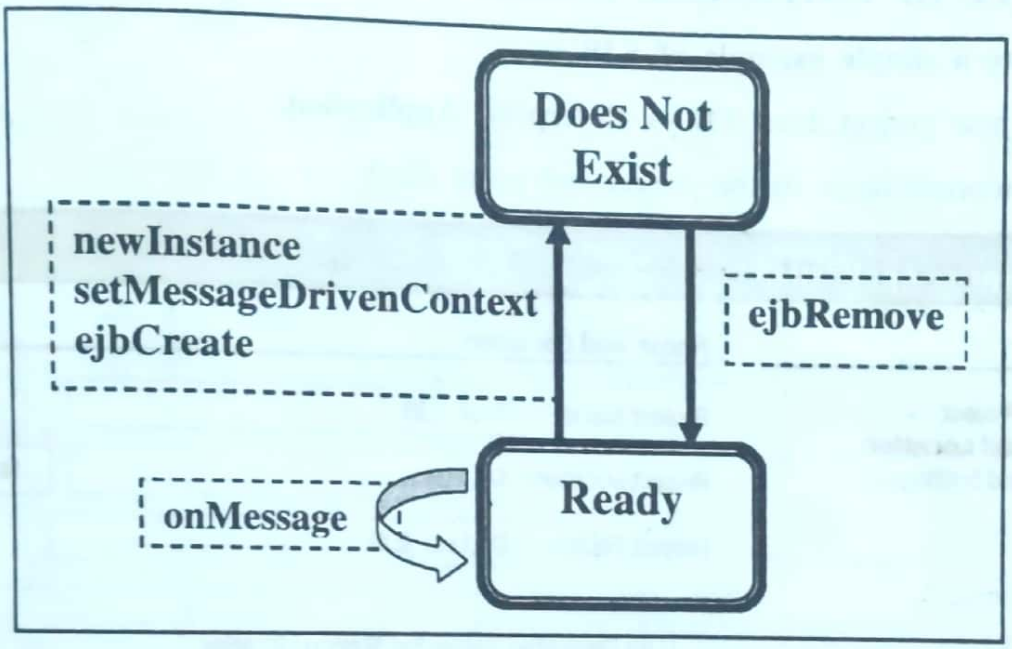
The Use of the Entity Bean :

You could probably use an entity bean under the following conditions :

- The bean represents a business entity and not a procedure.
- The bean's state must be persistent. If the bean instance terminates or if the Application Server is shut down, the bean's state still exists in persistent storage (a database).

3. Message Driven Bean :

Message driven beans are similar to session beans. They are actions. Message Driven Beans are called only when they receive some message. It asynchronous messages and it cannot be called for, directly by clients but it is activated by the container when a message arrives. Clients interact with these EJBs by sending messages to the queues. Following is the life cycle of Message Driven Bean (MDB) :



A message driven bean has the following two states :

1. **Does not exist :** In this state, the bean instance simply does not exist. Initially, the bean exists in the; does not exist state.
2. **Pooled state :** After invoking the ejbCreate() method, the MDB instance is in the ready pool, waiting to consume incoming messages. Since, MDBs are stateless, all instances of MDBs in the pool are identical; they're allocated to process a message and then return to the pool.

Methods for Message Driven Bean :

1. **onMessage (Message) :** This method is invoked for each message that is consumed by the bean. The container is responsible for serializing messages to a single message driven bean.
2. **ejbCreate( ) :** When this method is invoked, the MDB is first created and then, added to the 'to pool'.

3. **ejbCreate( ) :** When this method is invoked, the MDB is removed from the 'to pool'.
4. **setMessageDrivenContext(MessageDrivenContext) :** This method is called for, as a part of the event transition that message driven bean goes through, when it is being added to the pool. This is called for, just before the ejbCreate ( ).

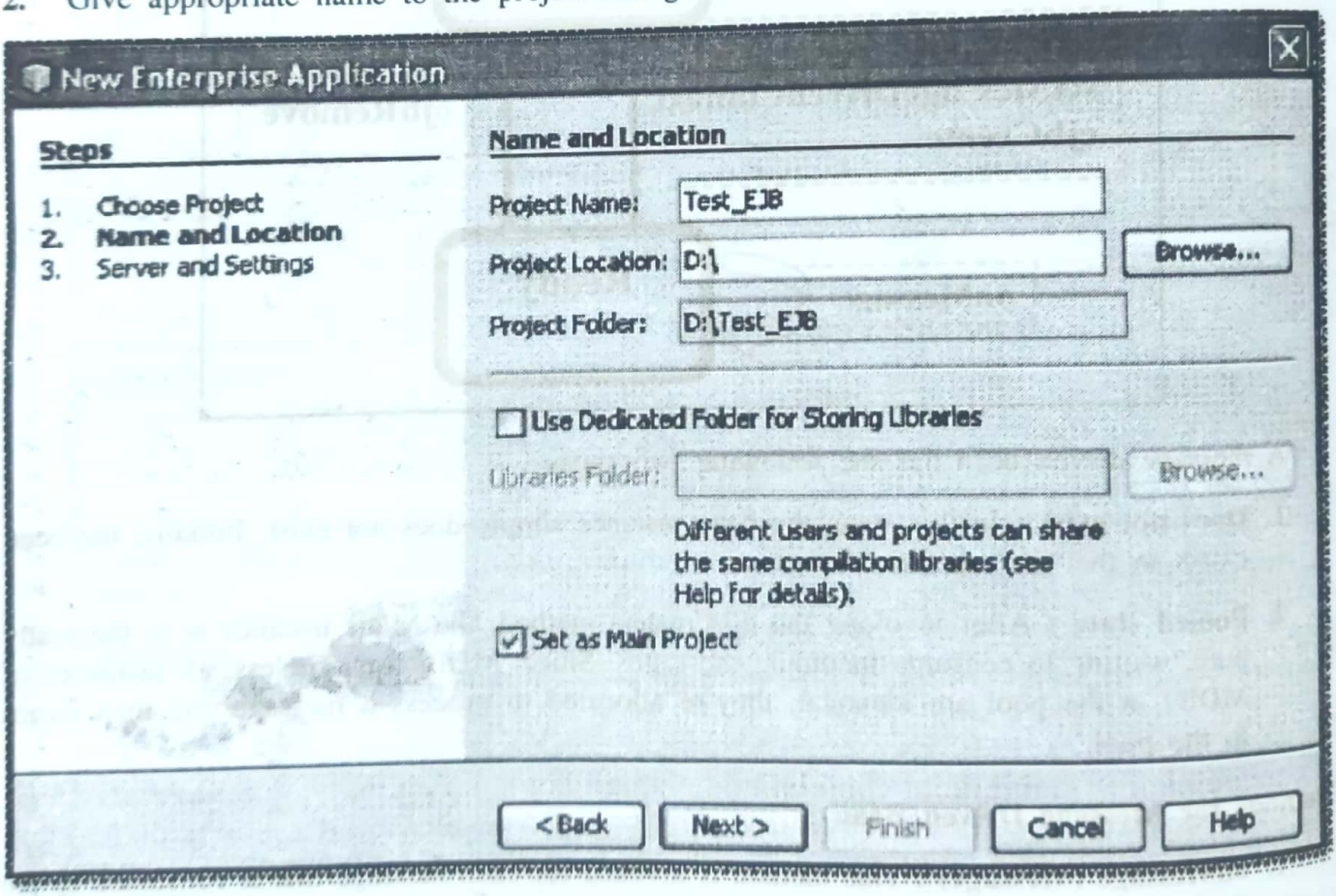
The Use of the Message Driven Bean :

Session beans allow you to send JMS messages and to receive them **synchronously**, but not **asynchronously**. To avoid tying up server resources, you may prefer not blocking synchronous receives in a server-side component. To receive messages asynchronously, use a message-driven bean.

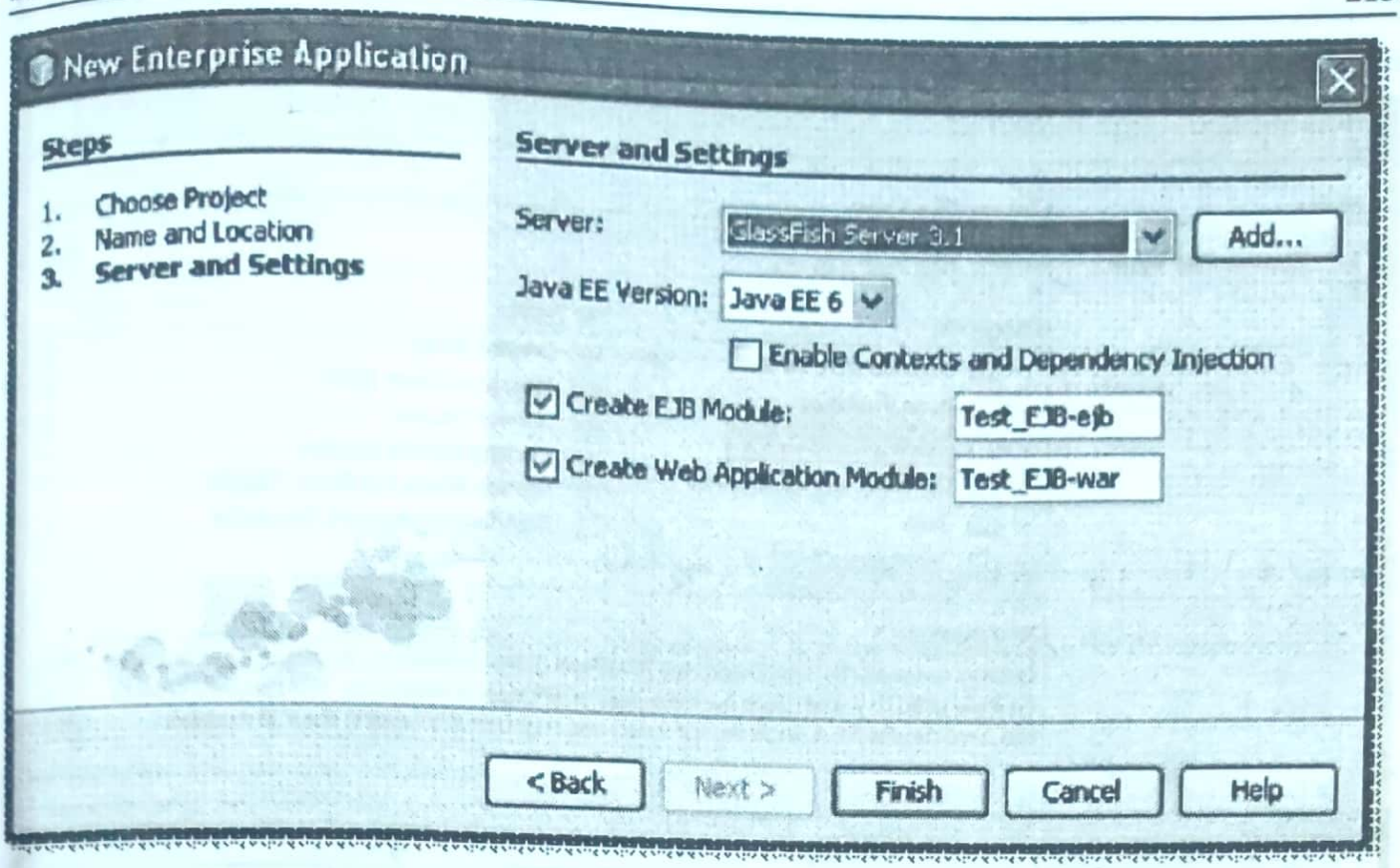
[8] EXAMPLE OF ENTERPRISE JAVABEANS

Steps to create a simple example of EJB :

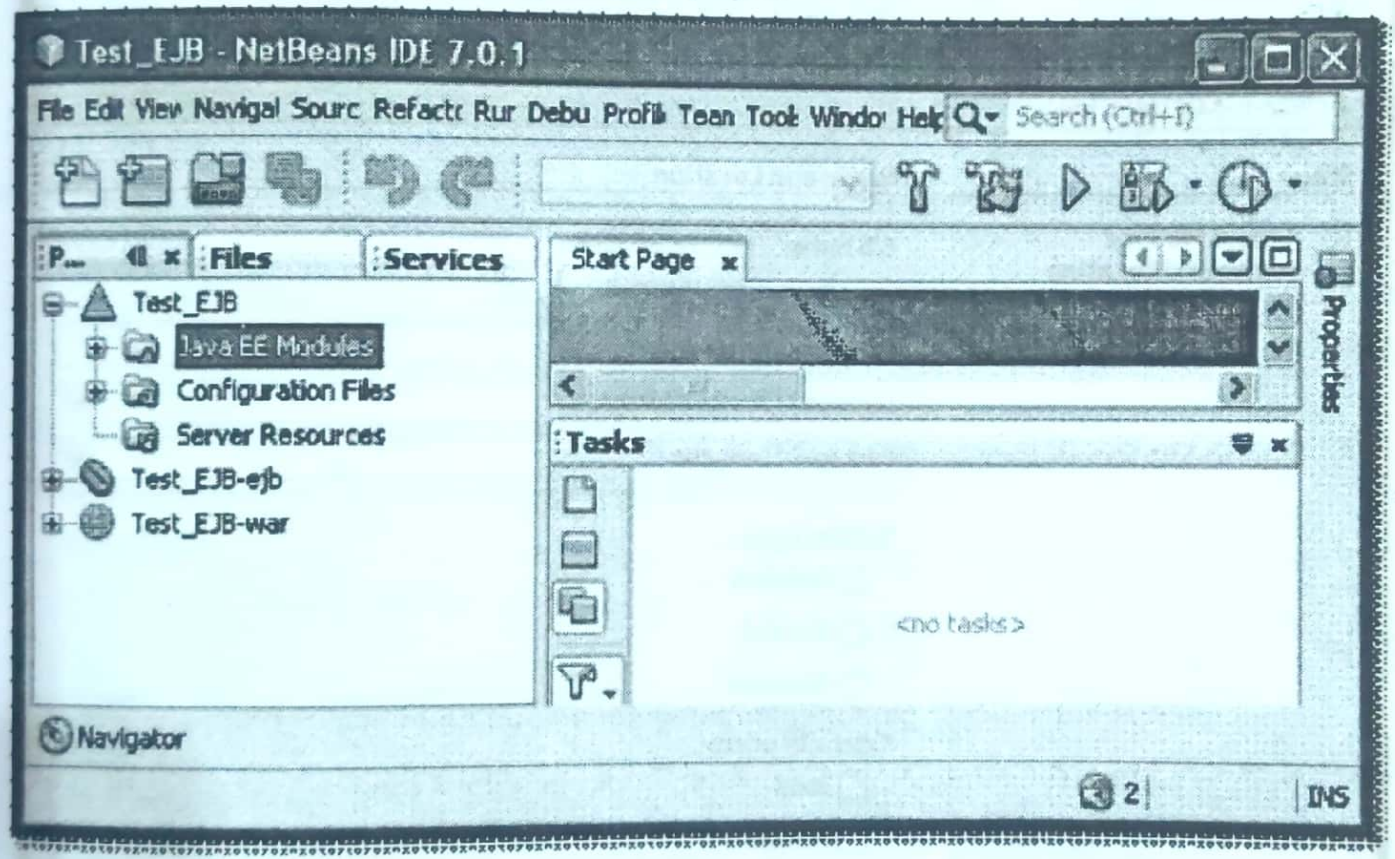
1. Create a new project **Java EE → Enterprise Application.**
2. Give appropriate name to the project and go to Next.



3. Select **Server and Version of J2EE** as shown and go to the Finish.

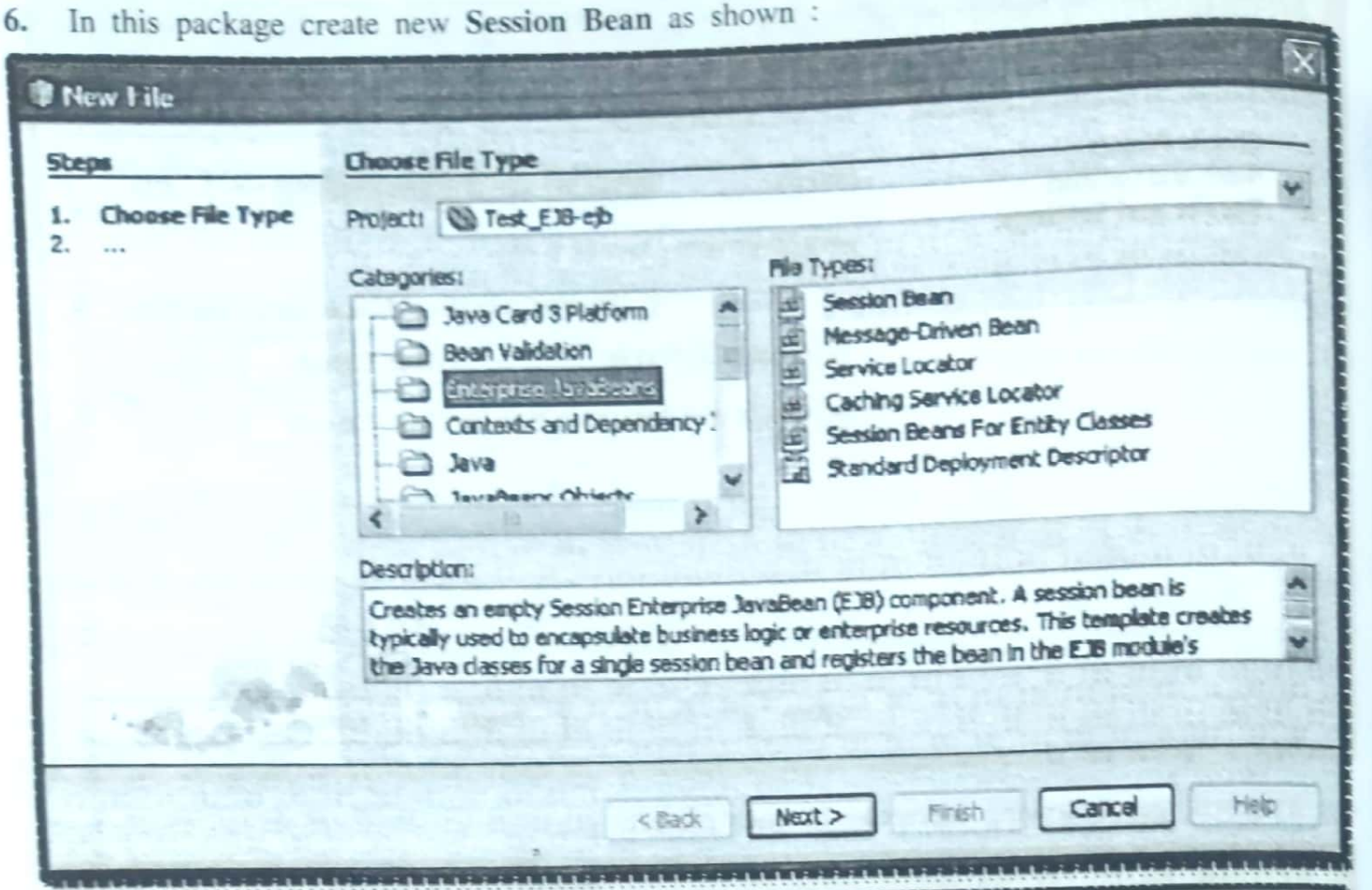


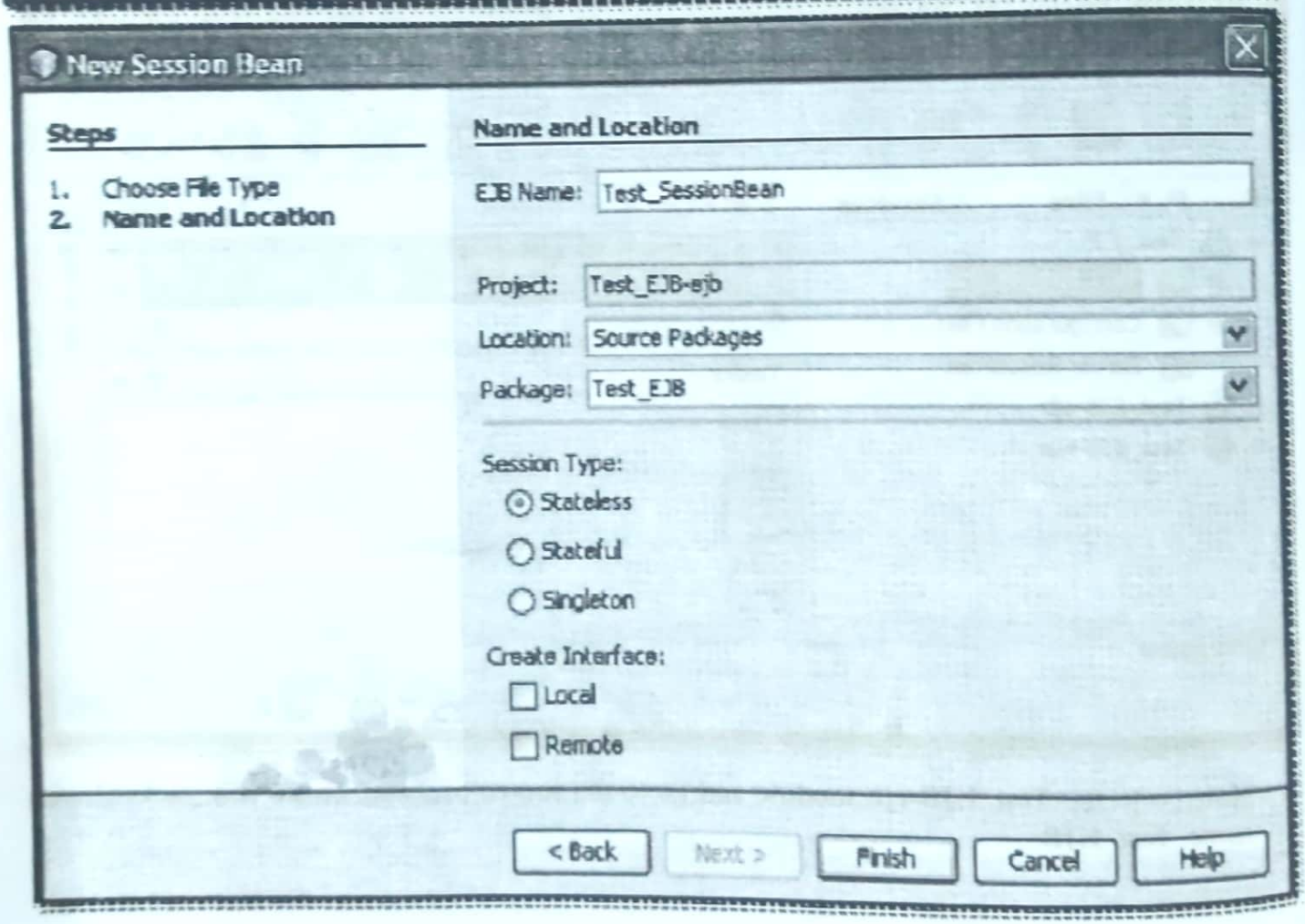
4. You will get following hierarchy in your project.



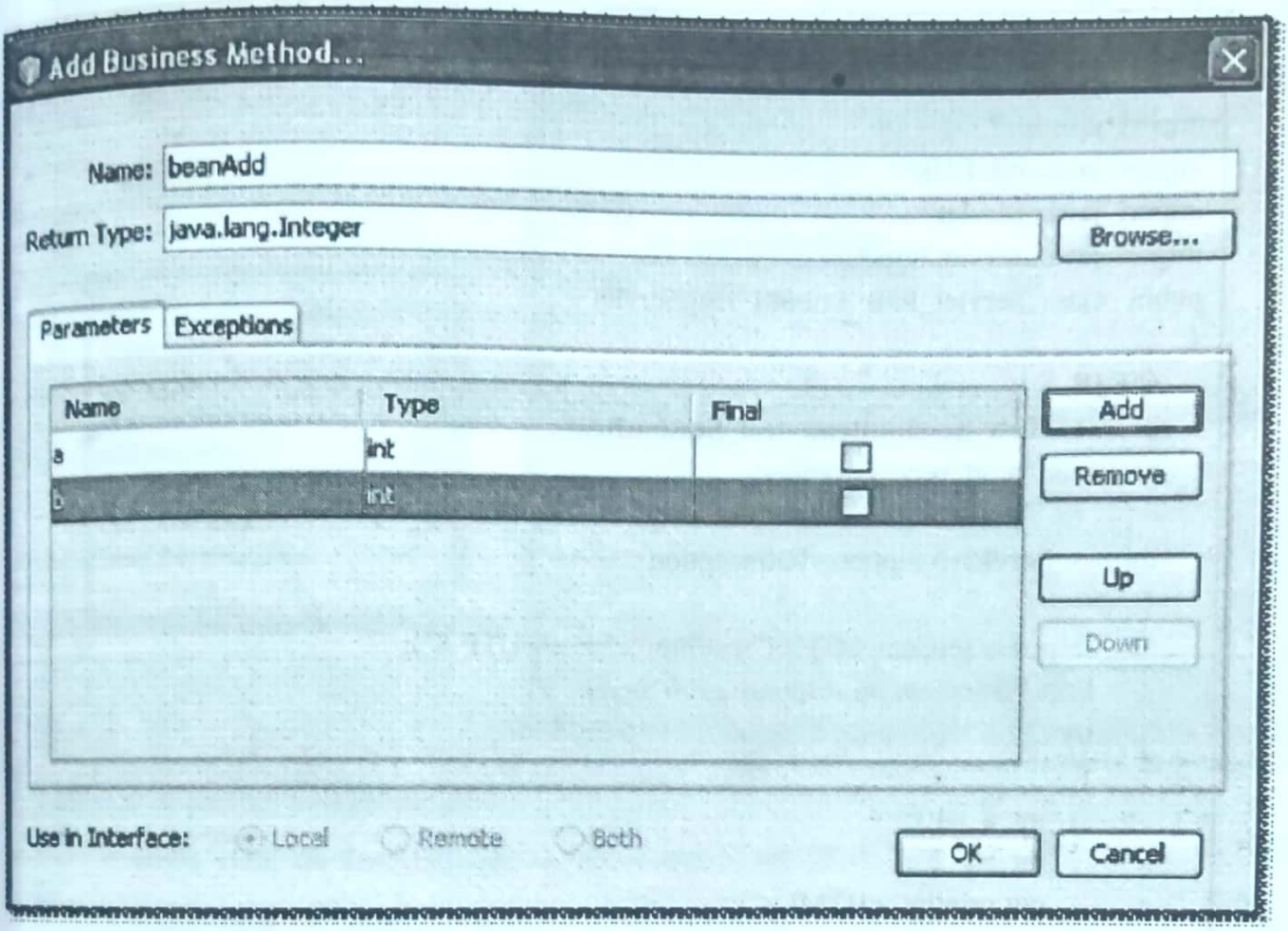
5. Now go to the **Test_EJB-ejb module** and go to the source package, create new package with name **Test_EJB**.

6. In this package create new **Session Bean** as shown :





7. Now right click on **Test_SessionBean.java** in to code window and select **Insert Code from menu**. Then select **Add Buisness Method from menu** as shown here :



8. Your code is as shown :

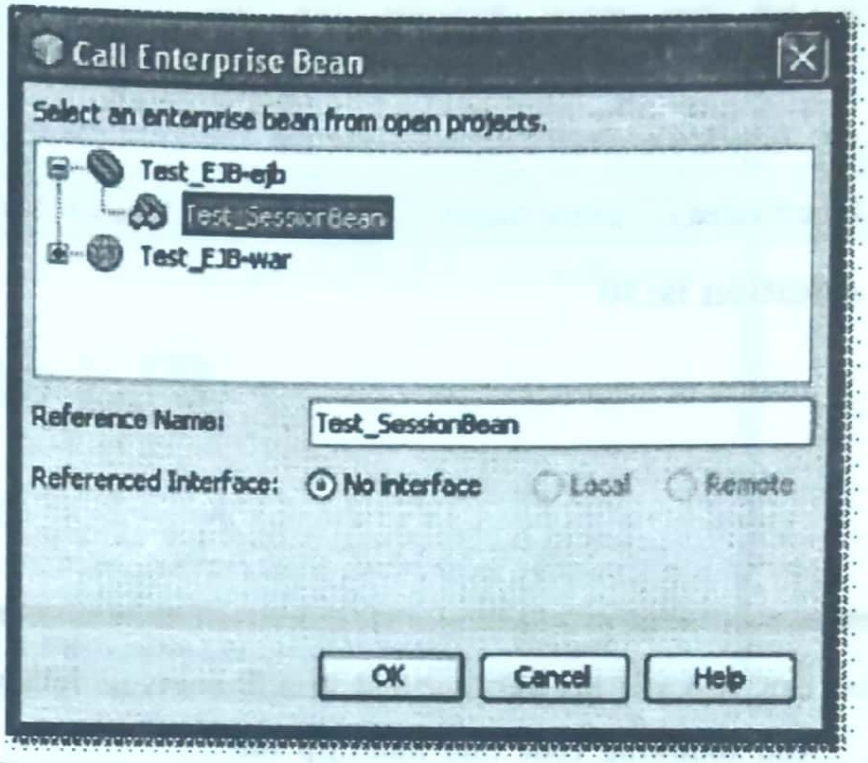
```
package Test_EJB;
import javax.ejb.Stateless;
import javax.ejb.LocalBean;
@Stateless
@LocalBean
public class Test_SessionBean
{
    public Integer beanAdd(int a, int b)
    {
        return a+b;
    }
}
```

9. Go to the **Test_EJB-war** and create new package **Test_EJB_EX** in source package. After that create a new Servlet with name **Servlet_EJB.java** with following code :

```
package Test_EJB_Ex;
import Test_EJB.Test_SessionBean;
import java.io.*;
import javax.ejb.EJB;
import javax.servlet.*;
import javax.servlet.http.*;
public class Servlet_EJB extends HttpServlet
{
   @EJB
   private Test_SessionBean test_SessionBean;
   protected void processRequest
                (HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
   {
       response.setContentType("text/html;charset=UTF-8");
       PrintWriter out = response.getWriter();
       try
       {
           int a =10;
           int b= 20;
           out.println("<HTML>");
           out.println("<Body>");
           out.println("<h3>");
           out.println(">Addition is:"+ String.valueOf(test_SessionBean.beanAdd(a, b)));
           out.println("</h3>");
           out.println("</Body>");
           out.println("</HTML>");
       }
       finally { out.close(); }
   }
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {processRequest(request, response);}
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {processRequest(request, response);}
}
```

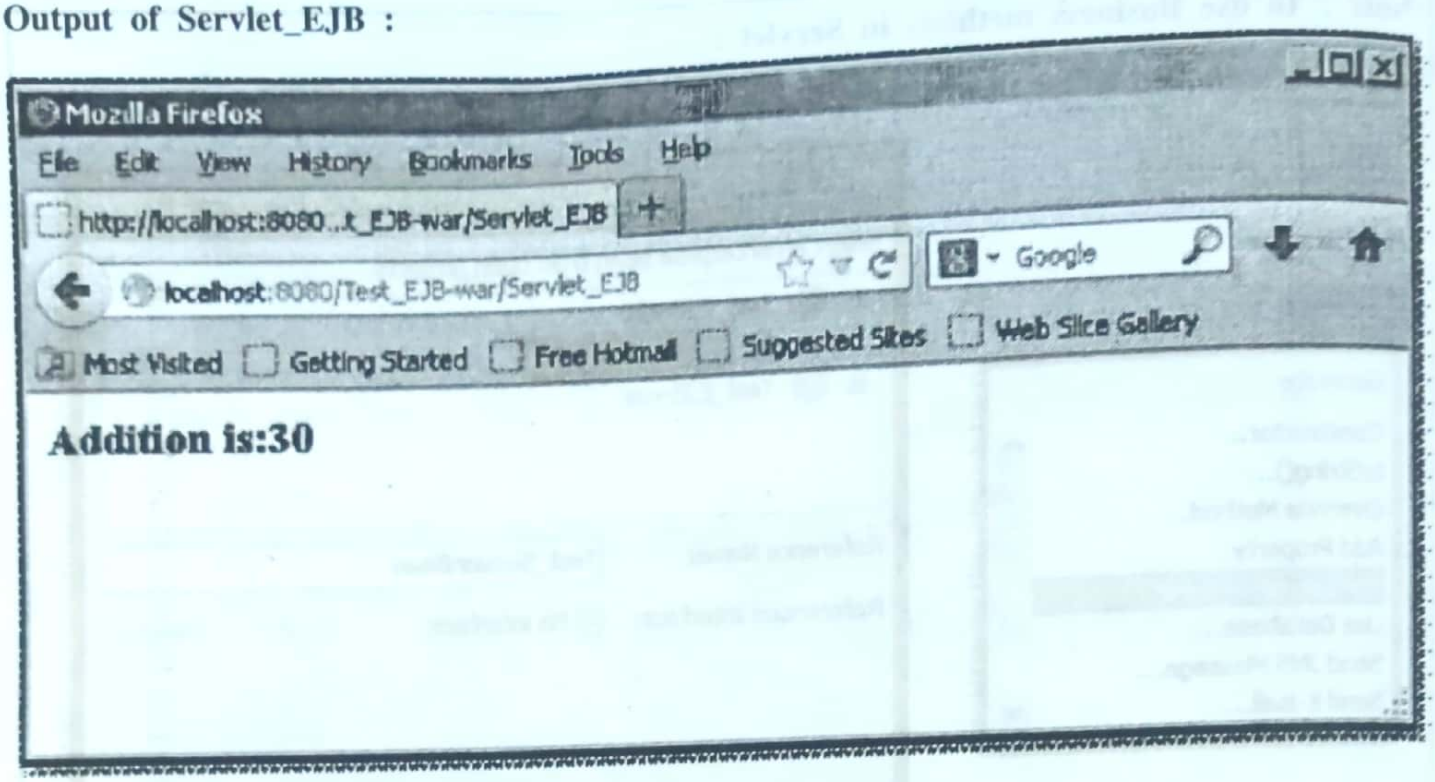
Note : To use Business methods in Servlet :

Right click on Servlet → Insert code → Call Enterprise Bean and select Test_SessionBean



10. Do following to run project :
    - Right click on Test_EJB-ejb → clean and build
    - Right click on Test_EJB-war → clean and build
    - Right click on Test_EJB → clean and build
    - Run the Enterprise Application Test_EJB, will run Test_EJB-war module with index.jsp.
    - **In address bar add URL pattern of servlet i.e /Servlet_EJB**

Output of Servlet_EJB :



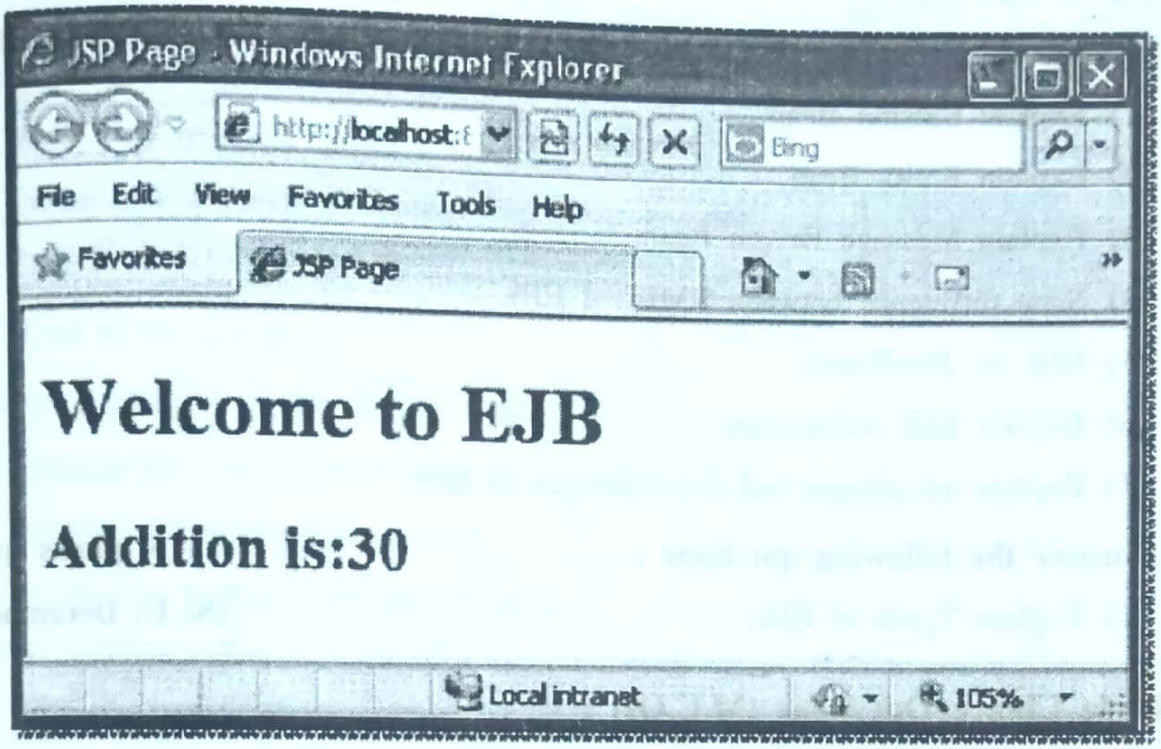
You can also use business method in JSP pages as follows :

1. **Add following code into index.jsp file :**

```
<%@page contentType="text/html" pageEncoding="UTF-8"
import="Test_EJB.Test_SessionBean" %>
<html>
   <head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
   <title>JSP Page</title>
   </head>
   <body>
   <h1>Welcome to EJB</h1>
   <%
       Test_SessionBean testSessionBean = new Test_SessionBean( );
   %>
   <h2>
   <%
       out.println("Addition is:"+String.valueOf(testSessionBean.beanAdd(10, 20)));
   %>
   </h2>
   </body>
</html>
```

2. **Build Test_EJB application and Run it.**

Output of index.jsp :



[9] SUMMARY

- EJB - is a reusable server-side software component.
- For EJB application an Application server (EJB Container) is required.
- EJB container provides services for Component Pooling, Resource Management, Transaction, Management, Security, Persistence and Handling of multiple clients.
- EJB architecture includes EJB clients, EJB home interface, EJB remote interface, EJB implementation and Container Implementation.
- Basic three types of Enterprise Java Beans is there :
  - o Session Bean
  - o Entity Beans
  - o Message-driven Beans
- A session EJB is a non persistent object.
- Session Bean is further divided into two types :
  - o Stateless
  - o Stateful
- Message Driven Beans are called only when they receive some message, otherwise it is similar to Session Bean.
- Entity EJBs represent persistent objects.
- Entity EJBs are always shared amongst clients.