What is EJB

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 RMI Tutorial 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.

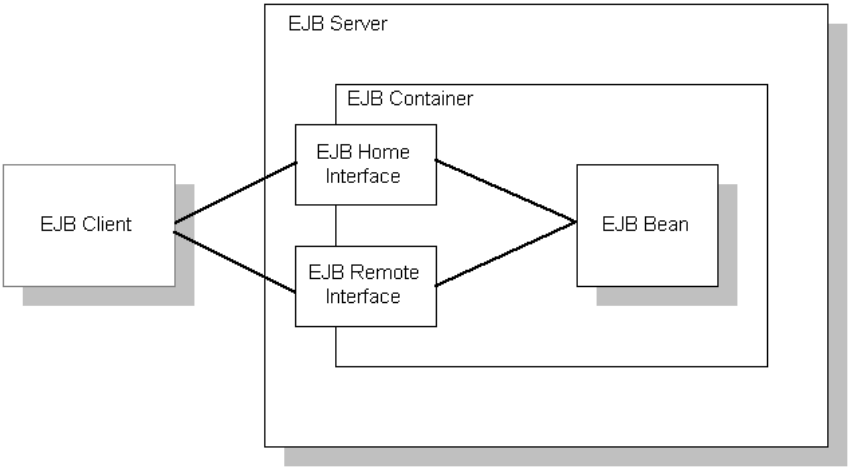
EJB application is deployed on the server, so it is called server side component also.

EJB is like COM (*Component Object Model*) provided by Microsoft. But, it is different from Java Bean, RMI and Web Services.

When use Enterprise Java Bean?

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.

Enterprise Bean Architecture



1. An Enterprise Bean Server
2. Enterprise Bean containers that run on these servers
3. Enterprise Beans that run in these containers
4. Enterprise Bean clients
5. Other systems such as the Java Naming and Directory interface (JNDI) and the Java Transaction Service(JTS).

How does the communication take place

- Client object makes a request for a method on the bean.
- Container comes into picture and checks whether the client is in the approved list for calling a method on the bean.
- If the client is authorized, container either creates new instance or activates the requested bean from the pool.
- Container ensures the bean gets its own new transaction
- Container informs the EJB object [wrapper class generated by container] that the bean is ready and passes the client's method request to the bean.

Enterprise Bean Server provides

- A Framework for creating, deploying and managing middle – tier business logic
- An environment that allows the execution of applications developed using EJB components.

In a three-tier environment:

- Client provides the User Interface logic
- The business rules are separated to the middle tier

- The database is the information repository

Client does not access the database directly. Instead, the client makes a call to the EJB server on the middle tier, which them accesses the database.

An EJB server takes care of:

- Managing and coordinating the allocation of resources to the applications.
- Security
- Threads
- Connection pooling
- Access to a distributed transaction management service.

- EJB server provides one or more containers for the enterprise beans, which is called the EJB container.
- An EJB container manages the enterprise beans contained within it.
- Enterprise beans are reusable modules of code that combine related tasks into a well-defined interface.
- These components are installed on an EJB server.

Types of Enterprise Java Bean

There are 3 types of enterprise bean in java.

Session Bean

Session bean contains business logic that can be invoked by local, remote or webservice client.

Message Driven Bean

Like Session Bean, it contains the business logic but it is invoked by passing message.

Entity Bean

It encapsulates the state that can be persisted in the database. It is deprecated. Now, it is replaced with JPA (Java Persistent API).

Disadvantages of EJB

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

Session Bean

Session bean encapsulates business logic only, it can be invoked by local, remote and webservice client.

It can be used for calculations, database access etc.

The life cycle of session bean is maintained by the application server (EJB Container).

Types of Session Bean

There are 3 types of session bean.

1) Stateless Session Bean: It doesn't maintain state of a client between multiple method calls.

2) Stateful Session Bean: It maintains state of a client across multiple requests.

3) Singleton Session Bean: One instance per application, it is shared between clients and supports concurrent access.

Stateless Session Bean

Stateless Session bean *is a business object that represents business logic* only. It doesn't have state (data).

In other words, *conversational state* between multiple method calls is not maintained by the container in case of stateless session bean.

The stateless bean objects are pooled by the EJB container to service the request on demand.

It can be accessed by one client at a time. In case of concurrent access, EJB container routes each request to different instance.

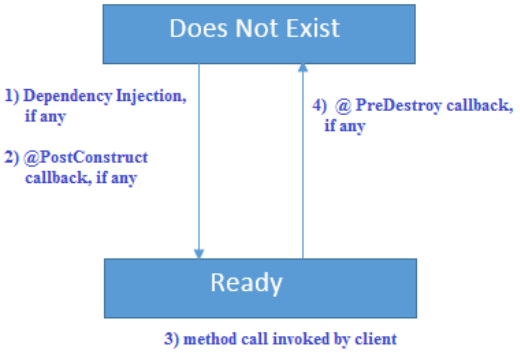
Annotations used in Stateless Session Bean

There are 3 important annotations used in stateless session bean:

1. @Stateless
2. @PostConstruct
3. @PreDestroy

Life cycle of Stateless Session Bean

There is only two states of stateless session bean: does not exist and ready. It is explained by the figure given below.



javaTpoint.com

EJB Container creates and maintains a pool of session bean first. It injects the dependency if then calls the @PostConstruct method if any. Now actual business logic method is invoked by the client. Then, container calls @PreDestory method if any. Now bean is ready for garbage collection.

Example of Stateless Session Bean

To develop stateless bean application, we are going to use **Eclipse IDE** and **glassfish 3** server.

To create EJB application, you need to create bean component and bean client.

1) Create stateless bean component

To create the stateless bean component, you need to create a remote interface and a bean class.

File: AdderImplRemote.java

```
1. package com.javatpoint;
2. import javax.ejb.Remote;
3.
4. @Remote
5. public interface AdderImplRemote {
6. int add(int a,int b);
7. }
```

File: AdderImpl.java

```
1. package com.javatpoint;
2. import javax.ejb.Stateless;
3.
4. @Stateless(mappedName="st1")
5. public class AdderImpl implements AdderImplRemote {
```

```
6.  public int add(int a,int b){
7.      return a+b;
8.  }
9. }
```

2) Create stateless bean client

The stateless bean client may be local, remote or webservice client. Here, we are going to create remote client. It is console based application. Here, we are not using dependency injection. The dependency injection can be used with web based client only.

File: AdderImpl.java

```
1. package com.javatpoint;
2. import javax.naming.Context;
3. import javax.naming.InitialContext;
4.
5. public class Test {
6. public static void main(String[] args)throws Exception {
7.     Context context=new InitialContext();
8.     AdderImplRemote remote=(AdderImplRemote)context.lookup("st1");
9.     System.out.println(remote.add(32,32));
10. }
11. }
```

Output

```
Output: 64
```

Stateful Session Bean

Stateful Session bean *is a business object that represents business logic* like stateless session bean. But, it maintains state (data).

In other words, *conversational state* between multiple method calls is maintained by the container in stateful session bean.

Annotations used in Stateful Session Bean

There are 5 important annotations used in stateful session bean:

1. @Stateful
2. @PostConstruct
3. @PreDestroy

4. @PrePassivate
5. @PostActivate

Example of Stateful Session Bean

To develop stateful session bean application, we are going to use **Eclipse IDE** and **glassfish 3** server.

As described in the previous example, you need to create bean component and bean client for creating session bean application.

1) Create stateful bean component

Let's create a remote interface and a bean class for developing stateful bean component.

File: BankRemote.java

```
1. package com.javatpoint;
2. import javax.ejb.Remote;
3. @Remote
4. public interface BankRemote {
5.     boolean withdraw(int amount);
6.     void deposit(int amount);
7.     int getBalance();
8. }
```

File: Bank.java

```
1. package com.javatpoint;
2. import javax.ejb.Stateful;
3. @Stateful(mappedName = "stateful123")
4. public class Bank implements BankRemote {
5.     private int amount=0;
6.     public boolean withdraw(int amount){
7.         if(amount<=this.amount){
8.             this.amount-=amount;
9.             return true;
10.         }else{
11.             return false;
12.         }
13.     }
14.     public void deposit(int amount){
15.         this.amount+=amount;
16.     }
```

```
17.     public int getBalance(){
18.         return amount;
19.     }
20. }
```

2) Create stateful bean client

The stateful bean client may be local, remote or webservice client. Here, we are going to create web based client and not using dependency injection.

File: index.jsp

```
1. <a href="OpenAccount">Open Account</a>
```

File: operation.jsp

```
1. <form action="operationprocess.jsp">
2. Enter Amount:<input type="text" name="amount"/><br>
3.
4. Choose Operation:
5. Deposit<input type="radio" name="operation" value="deposit"/>
6. Withdraw<input type="radio" name="operation" value="withdraw"/>
7. Check balance<input type="radio" name="operation" value="checkbalance"/>
8. <br>
9. <input type="submit" value="submit">
10. </form>
```

File: operationprocess.jsp

```
1. <%@ page import="com.javatpoint.*" %>
2. <%
3. BankRemote remote=(BankRemote)session.getAttribute("remote");
4. String operation=request.getParameter("operation");
5. String amount=request.getParameter("amount");
6.
7. if(operation!=null){
8.
9.     if(operation.equals("deposit")){
10.         remote.deposit(Integer.parseInt(amount));
11.         out.print("Amount successfully deposited!");
12.     }else
```

```
        if(operation.equals("withdraw")){
            boolean status=remote.withdraw(Integer.parseInt(amount));
            if(status){
                out.print("Amount successfully withdrawn!");
            }else{
                out.println("Enter less amount");
            }
        }else{
            out.println("Current Amount: "+remote.getBalance());
        }
}
%>
<hr/>
<jsp:include page="operation.jsp"></jsp:include>
```

File: OpenAccount.java

```
package com.javatpoint;
import java.io.IOException;
import javax.ejb.EJB;
import javax.naming.InitialContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/OpenAccount")
public class OpenAccount extends HttpServlet {
    //@EJB(mappedName="stateful123")
    //BankRemote b;
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        try{
            InitialContext context=new InitialContext();
            BankRemote b=(BankRemote)context.lookup("stateful123");

```

```
20.             request.getSession().setAttribute("remote",b);
21.             request.getRequestDispatcher("/operation.jsp").forward(request, response);
22.
23.         }catch(Exception e){System.out.println(e);}
24.     }
25. }
```