EJB

enterprise java beans

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Enterprise **J**ava **B**eans (EJB) is a development architecture for building highly scalable and robust enterprise level applications to be deployed on J2EE compliant Application Server such as JBOSS, Web Logic etc.

EJB 3.0 is being a great shift from EJB 2.0 and makes development of EJB based applications quite easy.

This tutorial is developed to provide a comprehensive understanding about the EJB concepts helpful to create and deploy an enterprise level application up and running.

Audience

This tutorial is designed for Software Professionals as well as for all those who are willing to learn EJB Programming. This tutorial will give you a great understanding about EJB Programming concepts in simple and easy steps.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of Java programming language, text editor, and execution of programs etc. Because we are going to develop enterprise-based applications using EJB, it will be good, if you have understanding on other technologies like Database Servers, Application Servers.

Copyright & Disclaimer

Table of Contents

1. EJB – OVERVIEW

EJB stands for **E**nterprise **J**ava **B**eans. EJB is an essential part of a J2EE platform. J2EE platform has component based architecture to provide multi-tiered, distributed and highly transactional features to enterprise level applications.

EJB provides an architecture to develop and deploy component based enterprise applications considering robustness, high scalability, and high performance. An EJB application can be deployed on any of the application server compliant with the J2EE 1.3 standard specification.

We'll be discussing EJB 3.0 in detail in this tutorial.

Types

EJB is primarily divided into three categories; following table lists their names with brief descriptions:

Type	Description
Session Bean	Session bean stores data of a particular user for a single session. It can be **stateful** or **stateless**. It is less resource intensive as compared to entity bean. Session bean gets destroyed as soon as user session terminates.
Entity Bean	**Entity beans** represent persistent data storage. User data can be saved to database via entity beans and later on can be retrieved from the database in the entity bean.
Message Driven Bean	**Message driven beans** are used in context of JMS (Java Messaging Service). Message Driven Beans can consumes JMS messages from external entities and act accordingly.

Benefits

Following are the important benefits of EJB:

- Simplified development of large-scale enterprise level application.
- Application Server/EJB container provides most of the system level services like transaction handling, logging, load balancing, persistence mechanism, exception handling, and so on. Developer has to focus only on business logic of the application.
- EJB container manages life cycle of EJB instances, thus developer needs not to worry about when to create/delete EJB objects.

2. EJB – ENVIRONMENT SETUP

EJB is a framework for Java, so the very first requirement is to have a **J**ava **D**evelopment **Kit** (JDK) installed in your machine.

System Requirement

JDK	1.5 or above.
Memory	no minimum requirement.
Disk Space	no minimum requirement.
Operating System	no minimum requirement.

Step 1 - Verify Java Installation in Your System

Now open console and execute the following **java** command.

OS	Task	Command
Windows	Open Command Console	c:\> java -version
Linux	Open Command Terminal	$ java -version
Mac	Open Terminal	machine:~ joseph$ java -version

Let's verify the output for all the operating systems:

OS	Output
Windows	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b11) Java HotSpot(TM) 64-Bit Server VM (build 23.21-b01, mixed mode)
Linux	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b11) Java HotSpot(TM) 64-Bit Server VM (build 23.21-b01, mixed mode)
Mac	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b11) Java HotSpot(TM) 64-Bit Server VM (build 23.21-b01, mixed mode)

If you do not have Java installed, install the Java Software Development Kit (SDK) from http://www.oracle.com/technetwork/java/javase/downloads/index.html.

We are assuming that Java 1.6.0_21 as installed version for this tutorial.

Step 2 – Set JAVA Environment

Set the **JAVA_HOME** environment variable to point the base directory location where Java is installed on your system. For example,

OS	Output
Windows	Set the environment variable JAVA_HOME to C:\Program Files\Java\jdk1.6.0_21
Linux	export JAVA_HOME=/usr/local/java-current
Mac	export JAVA_HOME=/Library/Java/Home

Append Java compiler location to System Path.

OS	Output
Windows	Append the string ;C:\Program Files\Java\jdk1.6.0_21\bin to the end of the system variable, Path.
Linux	export PATH=$PATH:$JAVA_HOME/bin/
Mac	not required

Verify Java Installation using **java -version** command explained above.

Step 3 – Download and Install NetBeans IDE

Download the latest version of NetBeans IDE from https://netbeans.org/downloads/index.html. At the time of writing this tutorial, I downloaded *Netbeans 7.3,* which comes bundled with JDK 1.7 using the following link http://www.oracle.com/technetwork/java/javase/downloads/index.html

OS	Installer name
Windows	Netbeans 7.3
Linux	Netbeans 7.3
Mac	Netbeans 7.3

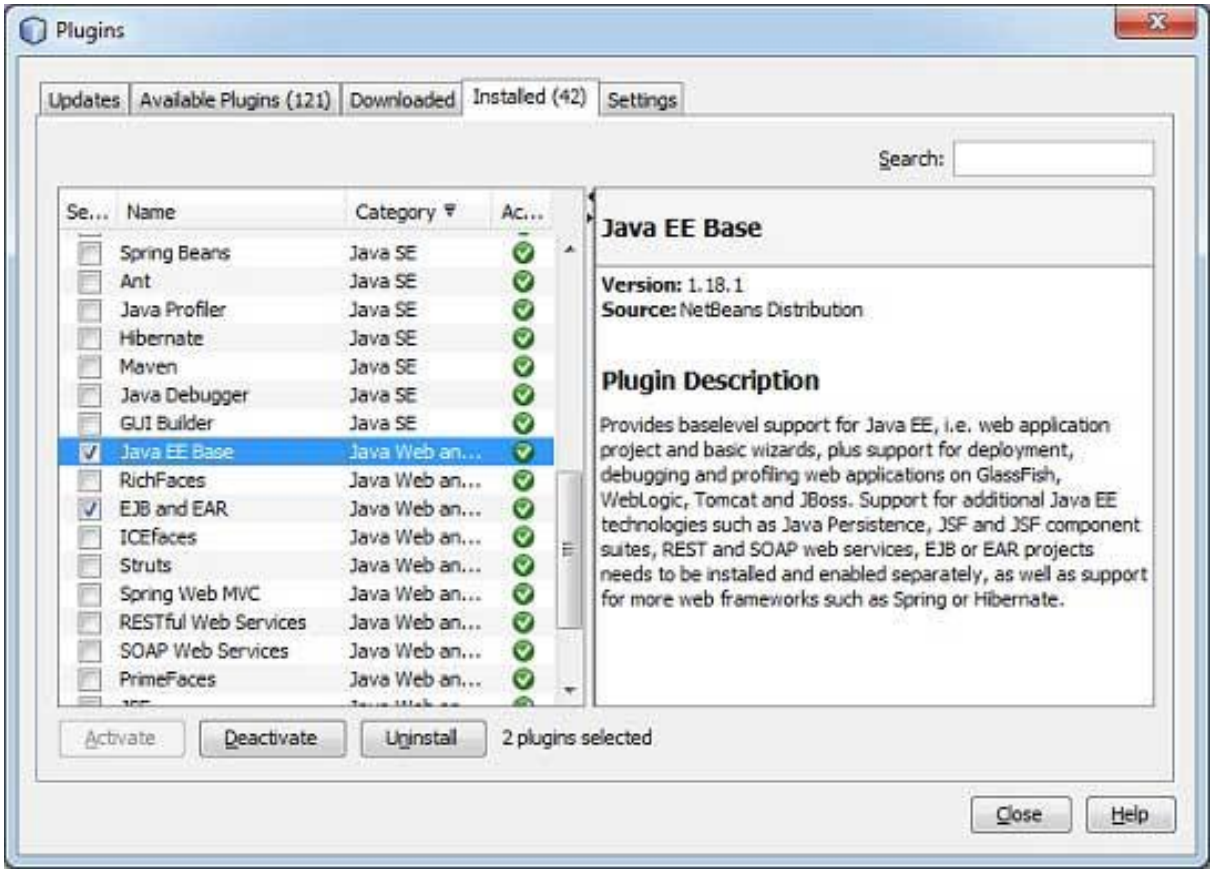
Step 4 – Set up JBoss Application Server

You can download the latest version of JBoss Server from http://www.jboss.org/jbossas/downloads/. Download the archive as per the platform. Extract the Jboss to any location on your machine.

OS	File name
Windows	jboss-5.1.0.GA-jdk6.zip
Linux	jboss-5.1.0.GA-src.tar.gz
Mac	jboss-5.1.0.GA-src.tar.gz

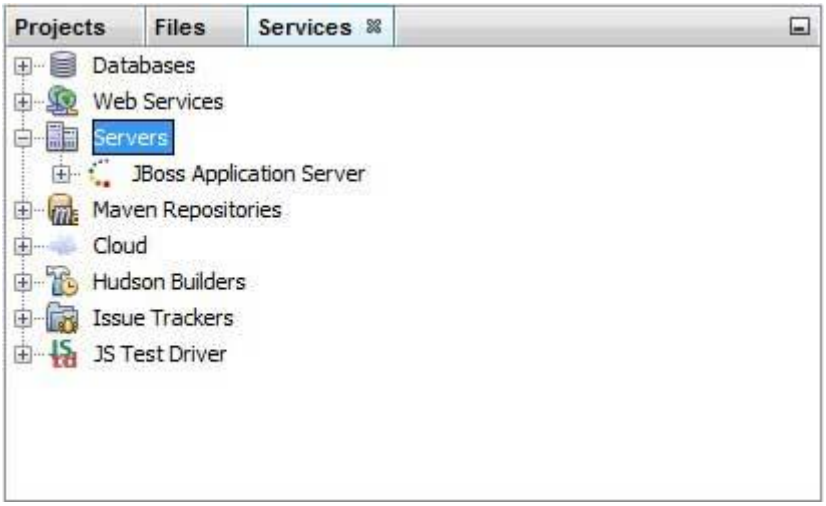
Step 5 – Configure JEE Plugins to Netbeans

Open Plugin window using Tools > Plugins. Open "Available Plugin" tab and select "Java EE Base" and "EJB and EAR" under "Java Web and EE" category. Click install button. Netbeans will download and install the respective plugins. Verify plugins installation using "Installed" tab (as shown in the image given below).

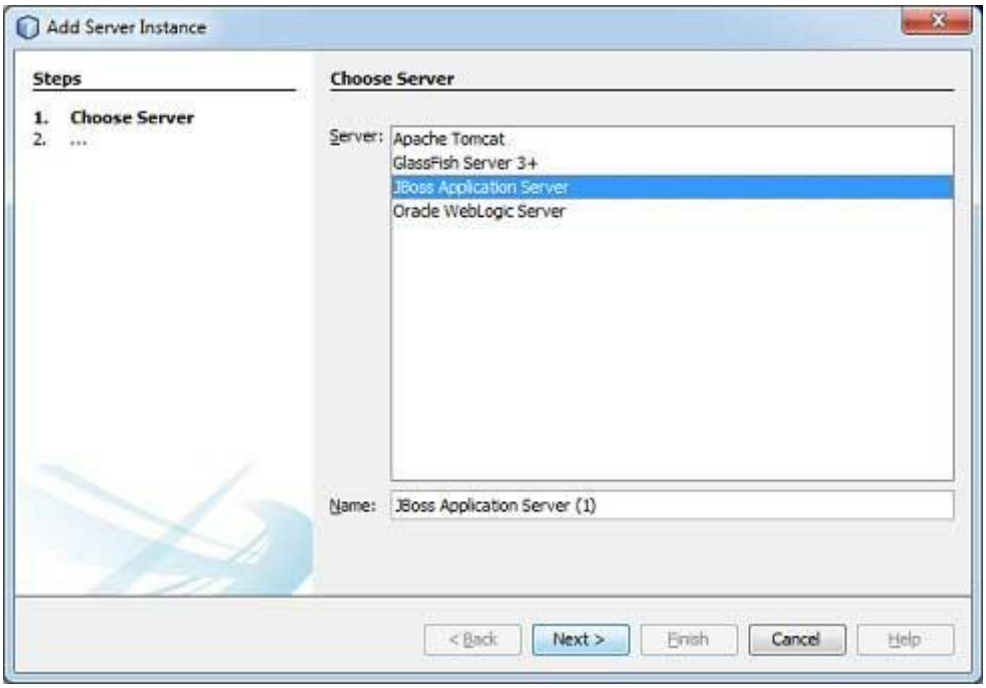


Step 6 – Configure JBoss Server in Netbeans

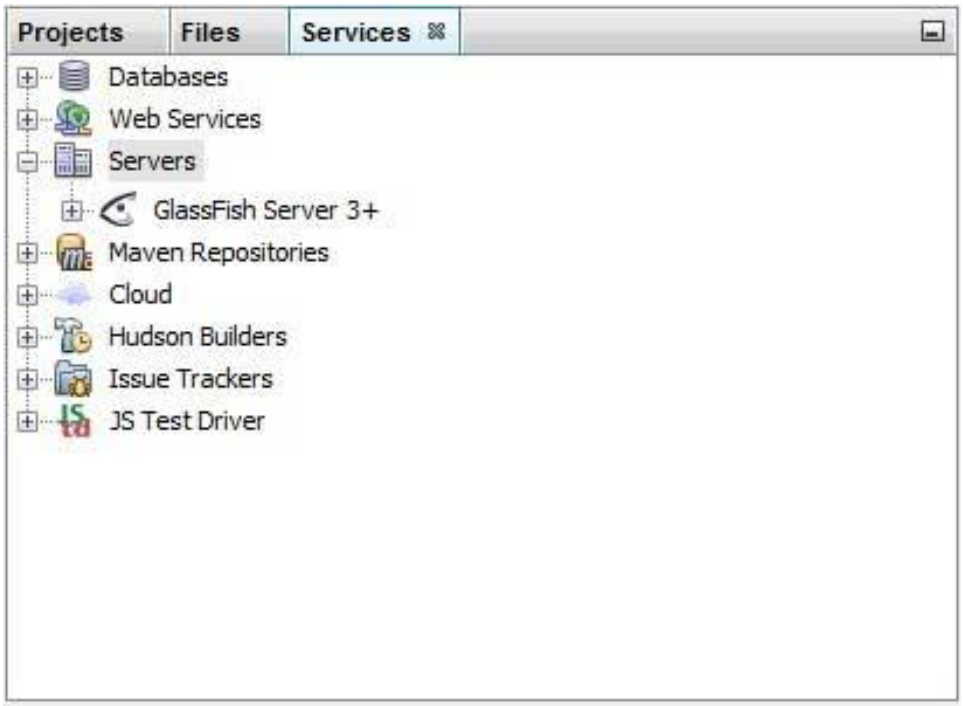
Go to Services tab and right click on servers to add a new server.



Add Server Instance wizard will open. Select JBoss and in next step enter the relevant details to configure server in netbeans.



Once everything is configured, you will see the following screen.



Step 7 – Install Database Server (PostGreSql)

Download latest version of PostGreSql database server from http://www.postgresql.org/download/. At the time of writing this tutorial, I downloaded *PostGreSql 9.2*

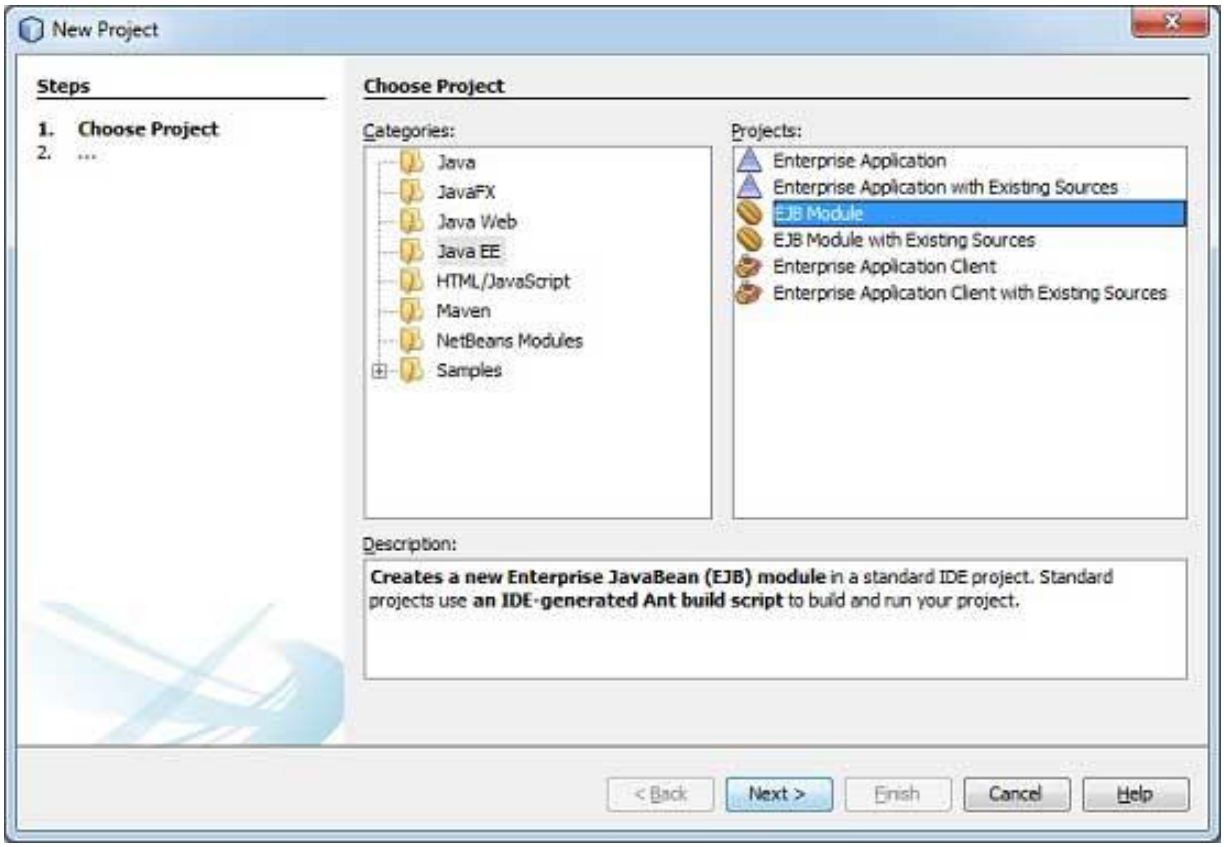
OS	Installer name
Windows	PostGreSql 9.2
Linux	PostGreSql 9.2
Mac	PostGreSql 9.2

3. EJB – CREATE APPLICATION

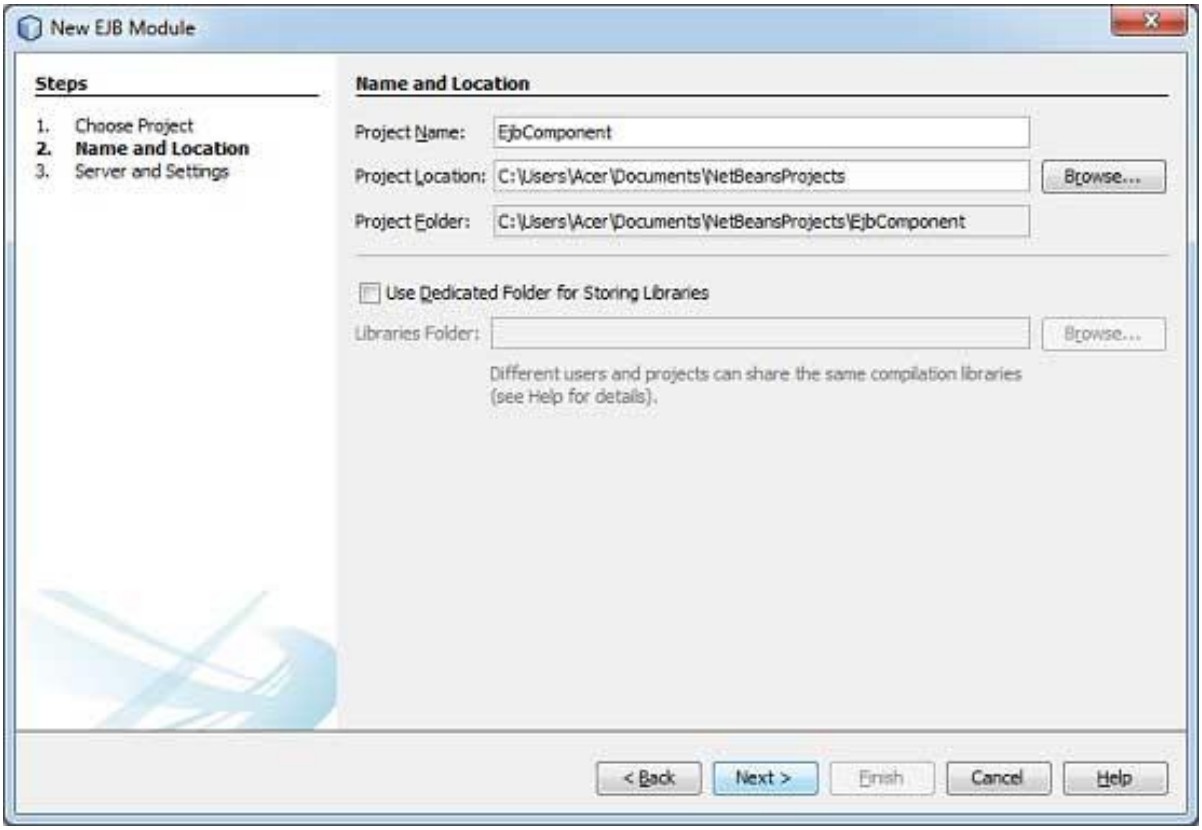
To create a simple EJB module, we will use NetBeans, "New project" wizard. In the example given below, We will create an EJB module project named Component.

Create Project

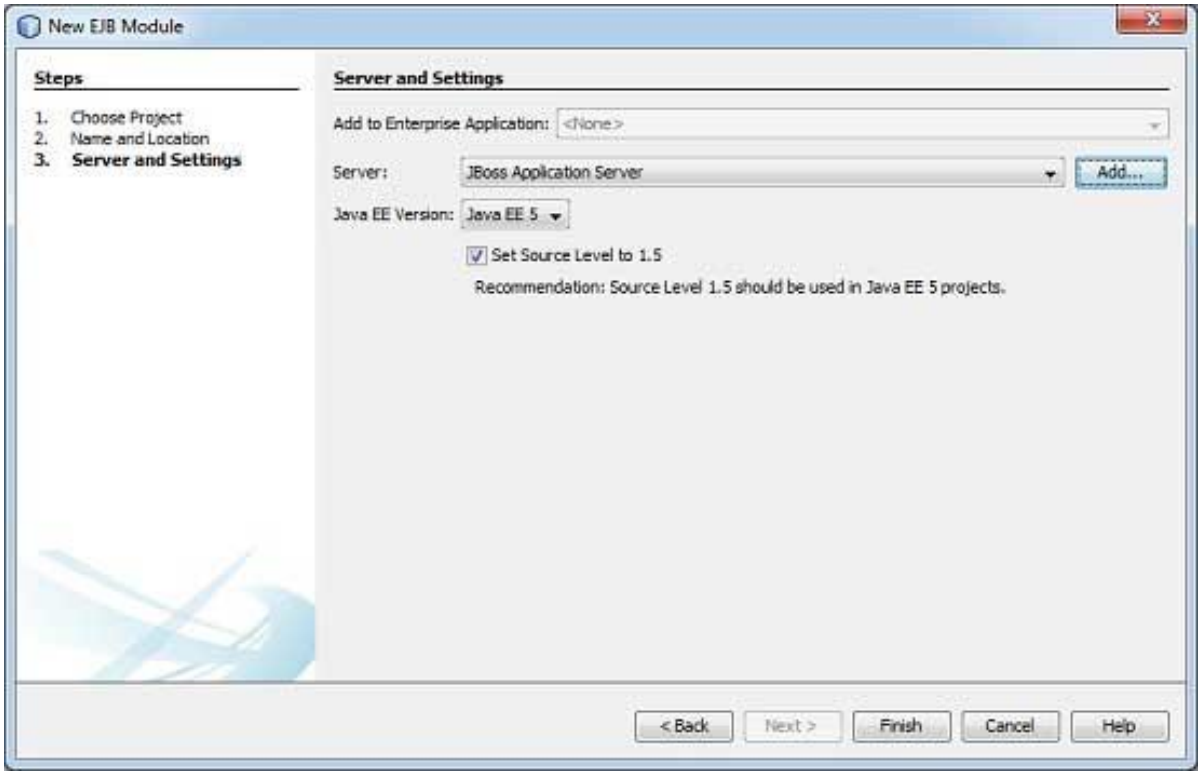
In NetBeans IDE, select **File > New Project >**. You will see the following screen.



Select project type under category **Java EE**, Project type as **EJB Module**. Click **Next >** button. You will see the following screen.



Enter project name and location. Click **Next >** button. You will see the following screen.

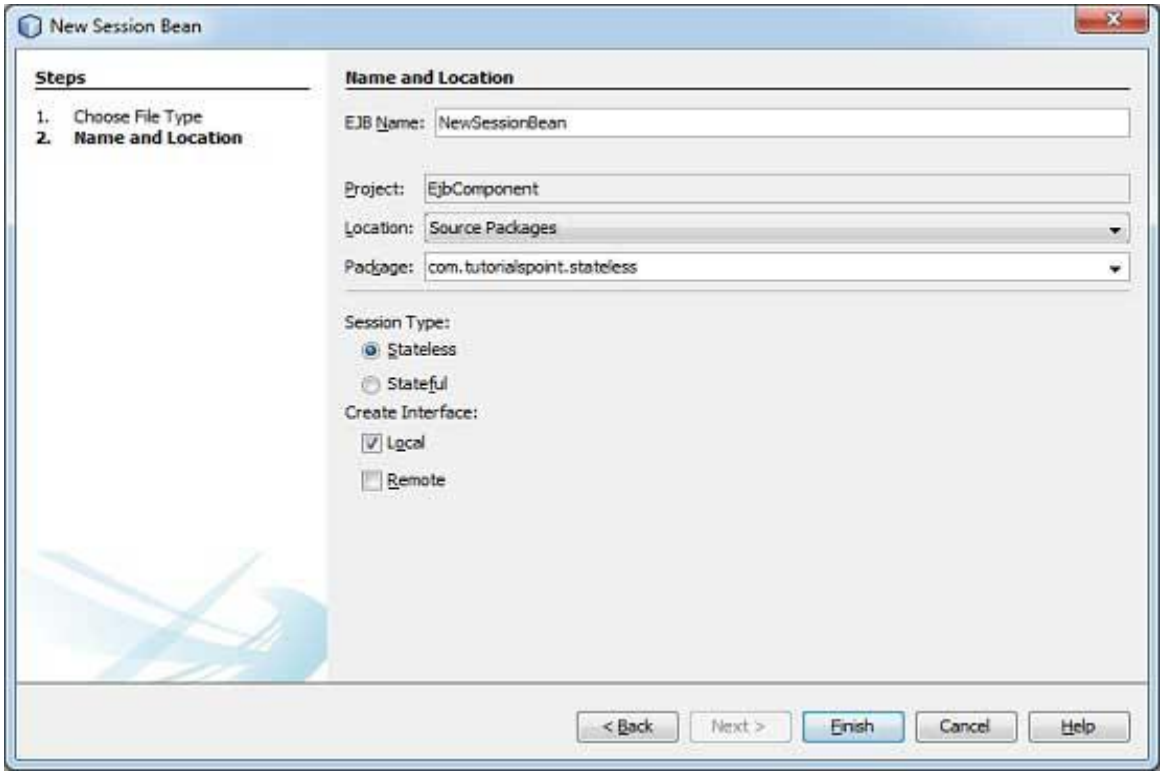


Select Server as **JBoss Application Server**. Click **Finish** button. You will see the following project created by NetBeans.

Create a Sample EJB

To create a simple EJB, we will use NetBeans "New" wizard. In the example given below, We will create a stateless EJB class named librarySessionBean under EjbComponent project.

Select project EjbComponent in project explorer window and right click on it. Select, **New > Session Bean**. You will see the **New Session Bean** wizard.



Enter session bean name and package name. Click **Finish** button. You will see the following EJB classes created by NetBeans.

- **LibrarySessionBean** - stateless session bean
- **LibrarySessionBeanLocal** - local interface for session bean

I am changing local interface to remote interface as we are going to access our EJB in a console based application. Remote/Local interface is used to expose business methods that an EJB has to implement.

LibrarySessionBeanLocal is renamed to LibrarySessionBeanRemote and LibrarySessionBean implements LibrarySessionBeanRemote interface.

LibrarySessionBeanRemote

```
package com.tutorialspoint.stateless;

import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibrarySessionBeanRemote {

    void addBook(String bookName);

    List getBooks();

}
```

LibrarySessionBean

```
package com.tutorialspoint.stateless;

import java.util.ArrayList;
import java.util.List;
import javax.ejb.Stateless;

@Stateless
public class LibrarySessionBean implements LibrarySessionBeanRemote {

    List<String> bookShelf;
```

```
    public LibrarySessionBean(){
       bookShelf = new ArrayList<String>();
    }

    public void addBook(String bookName) {
        bookShelf.add(bookName);
    }

    public List<String> getBooks() {
        return bookShelf;
    }
}
```

Build the Project

- Select EjbComponent project in Project Explorer window
- Right click on it to open context menu.
- Select clean and build.

You will see the following output in NetBeans console output.

```
ant -f C:\\EJB\\EjbComponent clean dist
init:
undeploy-clean:
deps-clean:
Deleting directory C:\EJB\EjbComponent\build
Deleting directory C:\EJB\EjbComponent\dist
clean:
init:
deps-jar:
Created dir: C:\EJB\EjbComponent\build\classes
Copying 3 files to C:\EJB\EjbComponent\build\classes\META-INF
Created dir: C:\EJB\EjbComponent\build\empty
Created dir: C:\EJB\EjbComponent\build\generated-sources\ap-source-output
Compiling 2 source files to C:\EJB\EjbComponent\build\classes
warning: [options] bootstrap class path not set in conjunction with -source 1.6
```

```
Note: C:\EJB\EjbComponent\src\java\com\tutorialspoint\stateless
\LibraryPersistentBean.java uses unchecked or unsafe operations.

Note: Recompile with -Xlint:unchecked for details.

1 warning
compile:
library-inclusion-in-archive:
Created dir: C:\EJB\EjbComponent\dist
Building jar: C:\EJB\EjbComponent\dist\EjbComponent.jar
dist:
BUILD SUCCESSFUL (total time: 3 seconds)
```

Start the Application Server

- Select JBoss application server under Servers in Services window.
- Right click on it to open context menu.
- Select start.

You will see the following output in NetBeans, output under JBoss Application Server.

```
Calling C:\jboss-5.1.0.GA\bin\run.conf.bat
=========================================================================

  JBoss Bootstrap Environment

  JBOSS_HOME: C:\jboss-5.1.0.GA

  JAVA: C:\Program Files (x86)\Java\jdk1.6.0_21\bin\java

  JAVA_OPTS: -Dprogram.name=run.bat -Xms128m -Xmx512m -server

  CLASSPATH: C:\jboss-5.1.0.GA\bin\run.jar

=========================================================================

16:25:50,062 INFO  [ServerImpl] Starting JBoss (Microcontainer)...
16:25:50,062 INFO  [ServerImpl] Release ID: JBoss [The Oracle] 5.1.0.GA (build:
SVNTag=JBoss_5_1_0_GA date=200905221634)
```

```
...


16:26:40,420 INFO  [TomcatDeployment] deploy, ctxPath=/admin-console

16:26:40,485 INFO  [config] Initializing Mojarra (1.2_12-b01-FCS) for context
'/admin-console'

16:26:42,362 INFO  [TomcatDeployment] deploy, ctxPath=/

16:26:42,406 INFO  [TomcatDeployment] deploy, ctxPath=/jmx-console

16:26:42,471 INFO  [Http11Protocol] Starting Coyote HTTP/1.1 on http-127.0.0.1-
8080

16:26:42,487 INFO  [AjpProtocol] Starting Coyote AJP/1.3 on ajp-127.0.0.1-8009

16:26:42,493 INFO  [ServerImpl] JBoss (Microcontainer) [5.1.0.GA (build:
SVNTag=JBoss_5_1_0_GA date=200905221634)] Started in 52s:427ms
```

Deploy the Project

- Select EjbComponent project in Project Explorer window.
- Right click on it to open context menu.
- Select Deploy.

You will see the following output in NetBeans console output.

```
ant -f C:\\EJB\\EjbComponent -DforceRedeploy=true -
Ddirectory.deployment.supported=false -Dnb.wait.for.caches=true run

init:

deps-jar:

compile:

library-inclusion-in-archive:

Building jar: C:\EJB\EjbComponent\dist\EjbComponent.jar

dist-directory-deploy:

pre-run-deploy:

Checking data source definitions for missing JDBC drivers...

Distributing C:\EJB\EjbComponent\dist\EjbComponent.jar to
[org.jboss.deployment.spi.LocalhostTarget@1e4f84ee]

Deploying C:\EJB\EjbComponent\dist\EjbComponent.jar

Application Deployed

Operation start started

Operation start completed

post-run-deploy:

run-deploy:
```

```
run:
BUILD SUCCESSFUL (total time: 2 seconds)
```

JBoss Application Server Log Output

```
16:30:00,963 INFO  [DeployHandler] Begin start, [EjbComponent.jar]

...

16:30:01,233 INFO  [Ejb3DependenciesDeployer] Encountered deployment
AbstractVFSDeploymentContext@12038795{vfszip:/C:/jboss-
5.1.0.GA/server/default/deploy/EjbComponent.jar/}

...

16:30:01,281 INFO  [JBossASKernel]    jndi:LibrarySessionBean/remote-
com.tutorialspoint.stateless.LibrarySessionBeanRemote

16:30:01,281 INFO  [JBossASKernel]
Class:com.tutorialspoint.stateless.LibrarySessionBeanRemote

16:30:01,281 INFO  [JBossASKernel]    jndi:LibrarySessionBean/remote

16:30:01,281 INFO  [JBossASKernel]  Added
bean(jboss.j2ee:jar=EjbComponent.jar,name=

LibrarySessionBean,service=EJB3) to KernelDeployment of: EjbComponent.jar

16:30:01,282 INFO  [JBossASKernel] installing bean:
jboss.j2ee:jar=EjbComponent.jar,name=BookMessageHandler,service=EJB3

16:30:01,282 INFO  [JBossASKernel]   with dependencies:

16:30:01,282 INFO  [JBossASKernel]   and demands:

16:30:01,282 INFO  [JBossASKernel]    jboss.ejb:service=EJBTimerService

...

16:30:01,283 INFO  [EJB3EndpointDeployer] Deploy
AbstractBeanMetaData@5497cb{name=jboss.j2ee:jar=EjbComponent.jar,
name=LibrarySessionBean, service=EJB3_endpoint
bean=org.jboss.ejb3.endpoint.deployers.impl.EndpointImpl properties=[container]
constructor=null autowireCandidate=true}

...

16:30:01,394 INFO  [SessionSpecContainer] Starting
jboss.j2ee:jar=EjbComponent.jar,name=LibrarySessionBean,service=EJB3

16:30:01,395 INFO  [EJBContainer] STARTED EJB:
com.tutorialspoint.stateless.LibrarySessionBean ejbName: LibrarySessionBean

16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:

   LibrarySessionBean/remote - EJB3.x Default Remote Business Interface

   LibrarySessionBean/remote-
com.tutorialspoint.stateless.LibrarySessionBeanRemote - EJB3.x Remote Business
Interface
```

```
16:30:02,723 INFO  [SessionSpecContainer] Starting
jboss.j2ee:jar=EjbComponent.jar,name=LibrarySessionBean,service=EJB3

16:30:02,723 INFO  [EJBContainer] STARTED EJB:
com.tutorialspoint.stateless.LibrarySessionBean ejbName: LibrarySessionBean
```

```
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:
```

```
   LibrarySessionBean/remote - EJB3.x Default Remote Business Interface

   LibrarySessionBean/remote-
com.tutorialspoint.stateless.LibrarySessionBeanRemote - EJB3.x Remote Business
Interface
```

Create Client to Access EJB

- In NetBeans IDE, select **File > New Project >**.
- Select project type under category **Java**, Project type as **Java Application**. Click **Next >** button.
- Enter project name and location. Click **Finish >** button. We have chosen name as EjbTester.
- Right click on project name in Project explorer window. Select **properties**.
- Add EJB component project created earlier under libraries using **Add Project** button in **compile** tab.
- Add jboss libraries using **Add jar/folder** button in **compile** tab. Jboss libraries can be located at <jboss installation folder>> client folder.

Create jndi.properties under project say EjbTester.

jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory

java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces

java.naming.provider.url=localhost
```

Create package com.tutorialspoint.test and EJBTester.java class under it.

EJBTester.java

```
package com.tutorialspoint.test;


import com.tutorialspoint.stateless.LibrarySessionBeanRemote;

import java.io.BufferedReader;
```

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;


public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
      props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
      ex.printStackTrace();
      }
      try {
      ctx = new InitialContext(props);
      } catch (NamingException ex) {
      ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }
   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testStatelessEjb();
   }
```

```
   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options \n1. Add Book\n2. Exit \nEnter Choice: ");
   }
   private void testStatelessEjb(){
      try {
         int choice = 1;
         LibrarySessionBeanRemote libraryBean =
         (LibrarySessionBeanRemote)ctx.lookup("LibrarySessionBean/remote");
         while (choice != 2) {
            String bookName;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               libraryBean.addBook(bookName);
            }else if (choice == 2) {
               break;
            }
         }
         List<String> booksList = libraryBean.getBooks();
         System.out.println("Book(s) entered so far: " + booksList.size());
         for (int i = 0; i < booksList.size(); ++i) {
         System.out.println((i+1)+". " + booksList.get(i));
         }
         LibrarySessionBeanRemote libraryBean1 =
         (LibrarySessionBeanRemote)ctx.lookup("LibrarySessionBean/remote");
         List<String> booksList1 = libraryBean1.getBooks();
         System.out.println(
         "***Using second lookup to get library stateless object***");
         System.out.println(
```

```
            "Book(s) entered so far: " + booksList1.size());
            for (int i = 0; i < booksList1.size(); ++i) {
               System.out.println((i+1)+". " + booksList1.get(i));
            }
         } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
         }finally {
            try {
               if(brConsoleReader !=null){
                  brConsoleReader.close();
               }
            } catch (IOException ex) {
               System.out.println(ex.getMessage());
            }
         }
      }
   }
```

Run Client to Access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 1
Enter book name: Learn Java
**********************
Welcome to Book Store
```

```
**********************
Options
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 1
1. Learn Java

***Using second lookup to get library stateless object***

Book(s) entered so far: 0
BUILD SUCCESSFUL (total time: 13 seconds)
```

In the following chapters, we will cover multiple aspects of this complete EJB application.

4. EJB – STATELESS BEAN

A stateless session bean is a type of enterprise bean, which is normally used to perform independent operations. A stateless session bean as per its name does not have any associated client state, but it may preserve its instance state. EJB Container normally creates a pool of few stateless bean's objects and use these objects to process client's request. Because of pool, instance variable values are not guaranteed to be same across lookups/method calls.

Steps to Create a Stateless EJB

Following are the steps required to create a stateless EJB:

- Create a remote/local interface exposing the business methods.
- This interface will be used by the EJB client application.
- Use @Local annotation, if EJB client is in same environment where EJB session bean is to be deployed.
- Use @Remote annotation, if EJB client is in different environment where EJB session bean is to be deployed.
- Create a stateless session bean, implementing the above interface.
- Use @Stateless annotation to signify it a stateless bean. EJB Container automatically creates the relevant configurations or interfaces required by reading this annotation during deployment.

Remote Interface

```
import javax.ejb.Remote;


@Remote
public interface LibrarySessionBeanRemote {
   //add business method declarations
}
```

Stateless EJB

```
@Stateless
public class LibrarySessionBean implements LibrarySessionBeanRemote {
   //implement business method
}
```

Example Application

Let us create a test EJB application to test stateless EJB.

Step	Description
1	Create a project with a name *EjbComponent* under a package *com.tutorialspoint.stateless* as explained in the *EJB - Create Application* chapter. You can also use the project created in *EJB - Create Application* chapter as such for this chapter to understand stateless EJB concepts.
2	Create *LibrarySessionBean.java* and *LibrarySessionBeanRemote* as explained in the *EJB - Create Application* chapter. Keep rest of the files unchanged.
3	Clean and Build the application to make sure business logic is working as per the requirements.
4	Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.
5	Now create the EJB client, a console based application in the same way as explained in the *EJB - Create Application* chapter under topic **Create Client to access EJB**.

EJBComponent (EJB Module)

LibrarySessionBeanRemote.java

```
package com.tutorialspoint.stateless;


import java.util.List;
import javax.ejb.Remote;


@Remote
public interface LibrarySessionBeanRemote {
   void addBook(String bookName);
   List getBooks();
}
```

LibrarySessionBean.java

```
package com.tutorialspoint.stateless;


import java.util.ArrayList;
```

```
import java.util.List;
import javax.ejb.Stateless;

@Stateless
public class LibrarySessionBean implements LibrarySessionBeanRemote {

   List<String> bookShelf;

   public LibrarySessionBean(){
      bookShelf = new ArrayList<String>();
   }

   public void addBook(String bookName) {
      bookShelf.add(bookName);
   }

   public List<String> getBooks() {
      return bookShelf;
   }
}
```

- As soon as you deploy the EjbComponent project on JBOSS, notice the jboss log.
- JBoss has automatically created a JNDI entry for our session bean - **LibrarySessionBean/remote**.
- We will be using this lookup string to get remote business object of type - **com.tutorialspoint.stateless.LibrarySessionBeanRemote**

JBoss Application Server Log Output

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:
   LibrarySessionBean/remote - EJB3.x Default Remote Business Interface
   LibrarySessionBean/remote-
com.tutorialspoint.stateless.LibrarySessionBeanRemote - EJB3.x Remote Business
Interface
16:30:02,723 INFO  [SessionSpecContainer] Starting
jboss.j2ee:jar=EjbComponent.jar,name=LibrarySessionBean,service=EJB3
```

```
16:30:02,723 INFO  [EJBContainer] STARTED EJB:
com.tutorialspoint.stateless.LibrarySessionBeanRemote ejbName:
LibrarySessionBean

16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:


   LibrarySessionBean/remote - EJB3.x Default Remote Business Interface

   LibrarySessionBean/remote-
com.tutorialspoint.stateless.LibrarySessionBeanRemote - EJB3.x Remote Business
Interface

...
```

EJBTester (EJB Client)

jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory

java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces

java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service.
- InitialContext object will be used to lookup stateless session bean.

EJBTester.java

```
package com.tutorialspoint.test;


import com.tutorialspoint.stateful.LibrarySessionBeanRemote;

import java.io.BufferedReader;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.List;

import java.util.Properties;

import javax.naming.InitialContext;

import javax.naming.NamingException;


public class EJBTester {
```

```
BufferedReader brConsoleReader = null;
Properties props;
InitialContext ctx;
{
   props = new Properties();
   try {
      props.load(new FileInputStream("jndi.properties"));
   } catch (IOException ex) {
      ex.printStackTrace();
   }
   try {
      ctx = new InitialContext(props);
   } catch (NamingException ex) {
      ex.printStackTrace();
   }
   brConsoleReader =
   new BufferedReader(new InputStreamReader(System.in));
}

public static void main(String[] args) {

   EJBTester ejbTester = new EJBTester();

   ejbTester.testStatelessEjb();
}

private void showGUI(){
   System.out.println("**********************");
   System.out.println("Welcome to Book Store");
   System.out.println("**********************");
   System.out.print("Options \n1. Add Book\n2. Exit \nEnter Choice: ");
}

private void testStatelessEjb(){

   try {
```

```
int choice = 1;

LibrarySessionBeanRemote libraryBean =
LibrarySessionBeanRemote)ctx.lookup("LibrarySessionBean/remote");

while (choice != 2) {
   String bookName;
   showGUI();
   String strChoice = brConsoleReader.readLine();
   choice = Integer.parseInt(strChoice);
   if (choice == 1) {
      System.out.print("Enter book name: ");
      bookName = brConsoleReader.readLine();
      Book book = new Book();
      book.setName(bookName);
      libraryBean.addBook(book);
   } else if (choice == 2) {
      break;
   }
}

List<Book> booksList = libraryBean.getBooks();

System.out.println("Book(s) entered so far: " + booksList.size());
int i = 0;
for (Book book:booksList) {
   System.out.println((i+1)+". " + book.getName());
   i++;
}
LibrarySessionBeanRemote libraryBean1 =
   (LibrarySessionBeanRemote)ctx.lookup("LibrarySessionBean/remote");
List<String> booksList1 = libraryBean1.getBooks();
System.out.println(
   "***Using second lookup to get library stateless object***");
System.out.println(
   "Book(s) entered so far: " + booksList1.size());
```

```
            for (int i = 0; i < booksList1.size(); ++i) {
               System.out.println((i+1)+". " + booksList1.get(i));
            }
         } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
         }finally {
            try {
               if(brConsoleReader !=null){
                  brConsoleReader.close();
               }
            } catch (IOException ex) {
               System.out.println(ex.getMessage());
            }
         }
      }
}
```

EJBTester performs the following tasks:

- Load properties from jndi.properties and initialize the InitialContext object.
- In testStatelessEjb() method, jndi lookup is done with name - "LibrarySessionBean/remote" to obtain the remote business object (stateless ejb).
- Then user is shown a library store User Interface and he/she is asked to enter choice.
- If user enters 1, the system asks for book name and saves the book using stateless session bean addBook() method. Session Bean is storing the book in its instance variable.
- If user enters 2, system retrieves books using stateless session bean getBooks() method and exits.
- Then another jndi lookup is done with the name - "LibrarySessionBean/remote" to obtain the remote business object (stateless EJB) again and listing of books is done.

Run Client to Access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
```

```
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 1
Enter book name: Learn Java
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 1
1. Learn Java
***Using second lookup to get library stateless object***
Book(s) entered so far: 0
BUILD SUCCESSFUL (total time: 13 seconds)
```

Run Client Again to Access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 0
```

```
***Using second lookup to get library stateless object***
Book(s) entered so far: 1
1. Learn Java
BUILD SUCCESSFUL (total time: 12 seconds)
```

- Output shown above may vary, depending upon how many stateless EJB object JBoss is maintaining.
- In case, a single stateless EJB object is maintained, you may see the same list of books after each lookup.
- EJB Container may return the same stateless EJB object for every lookup.
- Stateless EJB bean is keeping value of instance variable till the server is not restarted.

5. EJB – STATEFUL BEAN

A stateful session bean is a type of enterprise bean, which preserve the conversational state with client. A stateful session bean as per its name keeps associated client state in its instance variables. EJB Container creates a separate stateful session bean to process client's each request. As soon as request scope is over, stateful session bean is destroyed.

Steps to Create Stateful EJB

Following are the steps required to create a stateful EJB:

- Create a remote/local interface exposing the business methods.
- This interface will be used by the EJB client application.
- Use @Local annotation if EJB client is in same environment where EJB session bean need to be deployed.
- Use @Remote annotation if EJB client is in different environment where EJB session bean need to be deployed.
- Create a stateful session bean, implementing the above interface.
- Use @Stateful annotation to signify it a stateful bean. EJB Container automatically creates the relevant configurations or interfaces required by reading this annotation during deployment.

Remote Interface

```
import javax.ejb.Remote;


@Remote
public interface LibraryStatefulSessionBeanRemote {
   //add business method declarations
}
```

Stateful EJB

```
@Stateful
public class LibraryStatefulSessionBean implements
LibraryStatefulSessionBeanRemote {

   //implement business method

}
```

Example Application

Let us create a test EJB application to test stateful EJB.

Step	Description
1	Create a project with a name *EjbComponent* under a package *com.tutorialspoint.stateful* as explained in the *EJB - Create Application* chapter. You can also use the project created in *EJB - Create Application* chapter as such for this chapter to understand stateful EJB concepts.
2	Create *LibraryStatefulSessionBean.java* and *LibraryStatefulSessionBeanRemote* as explained in the *EJB - Create Application* chapter. Keep rest of the files unchanged.
3	Clean and Build the application to make sure business logic is working as per the requirements.
4	Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.
5	Now create the EJB client, a console based application in the same way as explained in the *EJB - Create Application* chapter under topic **Create Client to access EJB**.

EJBComponent (EJB Module)

LibraryStatefulSessionBeanRemote.java

```
package com.tutorialspoint.stateful;


import java.util.List;
import javax.ejb.Remote;


@Remote
public interface LibraryStatefulSessionBeanRemote {
   void addBook(String bookName);
   List getBooks();
}
```

LibraryStatefulSessionBean.java

```
package com.tutorialspoint.stateful;
import java.util.ArrayList;
import java.util.List;
import javax.ejb.Stateful;
```

```
@Stateful
public class LibraryStatefulSessionBean implements
LibraryStatefulSessionBeanRemote {

   List<String> bookShelf;

   public LibraryStatefulSessionBean(){
      bookShelf = new ArrayList<String>();
   }

   public void addBook(String bookName) {
      bookShelf.add(bookName);
   }

   public List<String> getBooks() {
      return bookShelf;
   }
}
```

- As soon as you deploy the EjbComponent project on JBOSS, notice the jboss log.
- JBoss has automatically created a JNDI entry for our session bean - **LibraryStatefulSessionBean/remote**.
- We will be using this lookup string to get remote business object of type - **com.tutorialspoint.stateful.LibraryStatefulSessionBeanRemote**

JBoss Application Server Log Output

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:
   LibraryStatefulSessionBean/remote - EJB3.x Default Remote Business Interface

   LibraryStatefulSessionBean/remote-
com.tutorialspoint.stateful.LibraryStatefulSessionBeanRemote - EJB3.x Remote
Business Interface

16:30:02,723 INFO  [SessionSpecContainer] Starting
jboss.j2ee:jar=EjbComponent.jar,name=LibraryStatefulSessionBean,service=EJB3
```

```
16:30:02,723 INFO  [EJBContainer] STARTED EJB:
com.tutorialspoint.stateful.LibraryStatefulSessionBeanRemote ejbName:
LibraryStatefulSessionBean

16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:


   LibraryStatefulSessionBean/remote - EJB3.x Default Remote Business Interface

   LibraryStatefulSessionBean/remote-
com.tutorialspoint.stateful.LibraryStatefulSessionBeanRemote - EJB3.x Remote
Business Interface

...
```

EJBTester (EJB Client)

jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory

java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces

java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service.
- InitialContext object will be used to lookup stateful session bean.

EJBTester.java

```
package com.tutorialspoint.test;


import com.tutorialspoint.stateful.LibraryStatefulSessionBeanRemote;

import java.io.BufferedReader;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.List;

import java.util.Properties;

import javax.naming.InitialContext;

import javax.naming.NamingException;


public class EJBTester {
```

```
BufferedReader brConsoleReader = null;
Properties props;
InitialContext ctx;
{
   props = new Properties();
   try {
      props.load(new FileInputStream("jndi.properties"));
   } catch (IOException ex) {
      ex.printStackTrace();
   }
   try {
      ctx = new InitialContext(props);
   } catch (NamingException ex) {
      ex.printStackTrace();
   }
   brConsoleReader =
   new BufferedReader(new InputStreamReader(System.in));
}

public static void main(String[] args) {

   EJBTester ejbTester = new EJBTester();

   ejbTester.testStatelessEjb();
}

private void showGUI(){
   System.out.println("**********************");
   System.out.println("Welcome to Book Store");
   System.out.println("**********************");
   System.out.print("Options \n1. Add Book\n2. Exit \nEnter Choice: ");
}


private void testStatelessEjb(){
```

```
      try {
         int choice = 1;

         LibraryStatefulSessionBeanRemote libraryBean =

LibraryStatefulSessionBeanRemote)ctx.lookup("LibraryStatefulSessionBean/remote"
);

         while (choice != 2) {
            String bookName;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               Book book = new Book();
               book.setName(bookName);
               libraryBean.addBook(book);
            } else if (choice == 2) {
               break;
            }
         }

         List<Book> booksList = libraryBean.getBooks();

         System.out.println("Book(s) entered so far: " + booksList.size());
         int i = 0;
         for (Book book:booksList) {
            System.out.println((i+1)+". " + book.getName());
            i++;
         }
         LibraryStatefulSessionBeanRemote libraryBean1 =

(LibraryStatefulSessionBeanRemote)ctx.lookup("LibraryStatefulSessionBean/remote
");
```

```
            List<String> booksList1 = libraryBean1.getBooks();
            System.out.println(
               "***Using second lookup to get library stateful object***");
            System.out.println(
               "Book(s) entered so far: " + booksList1.size());
            for (int i = 0; i < booksList1.size(); ++i) {
               System.out.println((i+1)+". " + booksList1.get(i));
            }
         } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
         }finally {
            try {
               if(brConsoleReader !=null){
                  brConsoleReader.close();
               }
            } catch (IOException ex) {
               System.out.println(ex.getMessage());
            }
         }
      }
   }
```

EJBTester performs the following tasks:

- Load properties from jndi.properties and initialize the InitialContext object.
- In testStatefulEjb() method, jndi lookup is done with name - "LibraryStatefulSessionBean/remote" to obtain the remote business object (stateful ejb).
- Then the user is shown a library store User Interface and he/she is asked to enter a choice.
- If user enters 1, system asks for book name and saves the book using stateful session bean addBook() method. Session Bean is storing the book in its instance variable.
- If user enters 2, system retrieves books using stateful session bean getBooks() method and exits.
- Then another jndi lookup is done with the name - "LibraryStatefulSessionBean/remote" to obtain the remote business object (stateful EJB) again and listing of books is done.

Run Client to Access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console:

```
run:
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 1
Enter book name: Learn Java
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 1
1. Learn Java
***Using second lookup to get library stateful object***
Book(s) entered so far: 0
BUILD SUCCESSFUL (total time: 13 seconds)
```

Run Client Again to Access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
**********************
```

```
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 0
***Using second lookup to get library stateful object***
Book(s) entered so far: 0
BUILD SUCCESSFUL (total time: 12 seconds)
```

- Output shown above states that for each lookup, a different stateful EJB instance is returned.
- Stateful EJB object is keeping value for single session only. As in second run, we are not getting any value of books.

6. EJB – PERSISTENCE

EJB 3.0, entity bean used in EJB 2.0 is largely replaced by persistence mechanism. Now entity bean is a simple POJO having mapping with table.

Following are the key actors in persistence API:

- **Entity** - A persistent object representing the data-store record. It is good to be serializable.
- **EntityManager** - Persistence interface to do data operations like add/delete/update/find on persistent object (entity). It also helps to execute queries using **Query** interface.
- **Persistence unit (persistence.xml)** - Persistence unit describes the properties of persistence mechanism.
- **Data Source (*ds.xml)** - Data Source describes the data-store related properties like connection url. user-name, password, etc.

To demonstrate EJB persistence mechanism, we need to do the following tasks:

- Step 1. Create table in database.
- Step 2. Create Entity class corresponding to table.
- Step 3. Create Data Source and Persistence Unit.
- Step 4. Create a stateless EJB having EntityManager instance.
- Step 5. Update stateless EJB. Add methods to add records and get records from database via entity manager.
- Step 6. A console based application client will access the stateless EJB to persist data in database.

Create Table

Create a table **books** in default database **postgres**.

```
CREATE TABLE books (
   id     integer PRIMARY KEY,
   name   varchar(50)
);
```

Create Entity Class

```
//mark it entity using Entity annotation

//map table name using Table annotation
@Entity
@Table(name="books")
public class Book implements Serializable{

   private int id;
   private String name;

   public Book(){
   }

   //mark id as primary key with autogenerated value
   //map database column id with id field
   @Id
   @GeneratedValue(strategy= GenerationType.IDENTITY)
   @Column(name="id")
   public int getId() {
      return id;
   }
   ...
}
```

Create DataSource and Persistence Unit

DataSource (jboss-ds.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
   <local-tx-datasource>
      <jndi-name>PostgresDS</jndi-name>
      <connection-url>jdbc:postgresql://localhost:5432/postgres</connection-url>
      <driver-class>org.postgresql.driver</driver-class>
      <user-name>sa</user-name>
      <password>sa</password>
```

```
      <min-pool-size>5</min-pool-size>
      <max-pool-size>20</max-pool-size>
      <idle-timeout-minutes>5</idle-timeout-minutes>
   </local-tx-datasource>
</datasources>
```

Persistence Unit (persistence.xml)

```
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
   <persistence-unit name="EjbComponentPU" transaction-type="JTA">
      <jta-data-source>java:/PostgresDS</jta-data-source>
      <exclude-unlisted-classes>false</exclude-unlisted-classes>
      <properties/>
   </persistence-unit>
   <persistence-unit name="EjbComponentPU2" transaction-type="JTA">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <jta-data-source>java:/PostgresDS</jta-data-source>
      <exclude-unlisted-classes>false</exclude-unlisted-classes>
      <properties>
         <property name="hibernate.hbm2ddl.auto" value="update"/>
      </properties>
   </persistence-unit>
</persistence>
```

Create Stateless EJB having EntityManager Instance

```
@Stateless
public class LibraryPersistentBean implements LibraryPersistentBeanRemote {

   //pass persistence unit to entityManager.
   @PersistenceContext(unitName="EjbComponentPU")
   private EntityManager entityManager;

   public void addBook(Book book) {
```

```
        entityManager.persist(book);
    }

    public List<Book> getBooks() {
        return entityManager.createQuery("From Books").getResultList();
    }
    ...
}
```

After building the EJB module, we need a client to access the stateless bean, which we will be going to create in the next section.

Example Application

Let us create a test EJB application to test EJB persistence mechanism.

Step	Description
1	Create a project with a name *EjbComponent* under a package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. You can also use the project created in *EJB - Create Application* chapter as such for this chapter to understand EJB persistence concepts.
2	Create *Book.java* under package *com.tutorialspoint.entity* and modify it as shown below.
3	Create *LibraryPersistentBean.java* and *LibraryPersistentBeanRemote* as explained in the *EJB - Create Application* chapters and modify them as shown below.
4	Create *jboss-ds.xml* in **EjbComponent > setup** folder and *persistence.xml* in **EjbComponent > src > conf** folder. These folders can be seen in files tab in Netbeans. Modify these files as shown above.
5	Clean and Build the application to make sure business logic is working as per the requirements.
6	Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.
7	Now create the EJB client, a console based application in the same way as explained in the *EJB - Create Application* chapter under topic **Create Client to access EJB**. Modify it as shown below.

EJBComponent (EJB Module)

Book.java

```
package com.tutorialspoint.entity;
```

```
import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EntityListeners;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="books")
public class Book implements Serializable{

   private int id;
   private String name;

   public Book(){
   }

   @Id
   @GeneratedValue(strategy= GenerationType.IDENTITY)
   @Column(name="id")
   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }
```

```
   public void setName(String name) {
      this.name = name;
   }
}
```

LibraryPersistentBeanRemote.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryPersistentBeanRemote {

   void addBook(Book bookName);

   List<Book> getBooks();

}
```

LibraryPersistentBean.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
public class LibraryPersistentBean implements LibraryPersistentBeanRemote {

   public LibraryPersistentBean(){
   }
```

```
   @PersistenceContext(unitName="EjbComponentPU")
   private EntityManager entityManager;

   public void addBook(Book book) {
      entityManager.persist(book);
   }

   public List<Book> getBooks() {
      return entityManager.createQuery("From Book").getResultList();
   }
}
```

- As soon as you deploy the EjbComponent project on JBOSS, notice the jboss log.
- JBoss has automatically created a JNDI entry for our session bean - **LibraryPersistentBean/remote**.
- We will be using this lookup string to get remote business object of type - **com.tutorialspoint.stateless.LibraryPersistentBeanRemote**

JBoss Application Server Log Output

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:
   LibraryPersistentBean/remote - EJB3.x Default Remote Business Interface
   LibraryPersistentBean/remote-
com.tutorialspoint.stateless.LibraryPersistentBeanRemote - EJB3.x Remote
Business Interface
16:30:02,723 INFO  [SessionSpecContainer] Starting
jboss.j2ee:jar=EjbComponent.jar,name=LibraryPersistentBeanRemote,service=EJB3
16:30:02,723 INFO  [EJBContainer] STARTED EJB:
com.tutorialspoint.stateless.LibraryPersistentBeanRemote ejbName:
LibraryPersistentBean
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:

   LibraryPersistentBean/remote - EJB3.x Default Remote Business Interface
   LibraryPersistentBean/remote-
com.tutorialspoint.stateless.LibraryPersistentBeanRemote - EJB3.x Remote
Business Interface
```

```
...
```

EJBTester (EJB Client)

jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service.
- InitialContext object will be used to lookup stateless session bean.

EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
```

```
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testEntityEjb();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options \n1. Add Book\n2. Exit \nEnter Choice: ");
   }

   private void testEntityEjb(){

      try {
         int choice = 1;

         LibraryPersistentBeanRemote libraryBean =

LibraryPersistentBeanRemote)ctx.lookup("LibraryPersistentBean/remote");

         while (choice != 2) {
            String bookName;
```

```
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);

            if (choice == 1) {

               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               Book book = new Book();
               book.setName(bookName);
               libraryBean.addBook(book);
            } else if (choice == 2) {
               break;
            }
         }

         List<Book> booksList = libraryBean.getBooks();

         System.out.println("Book(s) entered so far: " + booksList.size());
         int i = 0;
         for (Book book:booksList) {
            System.out.println((i+1)+". " + book.getName());
            i++;
         }
      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
   }
```

```
}
```

EJBTester performs the following tasks.

- Load properties from jndi.properties and initialize the InitialContext object.
- In testStatefulEjb() method, jndi lookup is done with name - "LibraryStatefulSessionBean/remote" to obtain the remote business object (stateful ejb).
- Then user is shown a library store User Interface and he/she is asked to enter choice.
- If user enters 1, system asks for book name and saves the book using stateless session bean addBook() method. Session Bean is persisting the book in database via EntityManager call.
- If user enters 2, system retrieves books using stateful session bean getBooks() method and exits.
- Then another jndi lookup is done with name - "LibraryStatelessSessionBean/remote" to obtain the remote business object (stateless EJB) again and listing of books is done.

Run Client to Access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console:

```
run:
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 1
Enter book name: Learn Java
**********************
Welcome to Book Store
**********************
Options
```

```
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 1
1. learn java
BUILD SUCCESSFUL (total time: 15 seconds)
```

Run Client Again to Access EJB

Restart the JBoss before accessing the EJB.

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console:

```
run:
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 1
Enter book name: Learn Spring
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 2
1. learn java
2. Learn Spring
BUILD SUCCESSFUL (total time: 15 seconds)
```

The output shown above states that books are getting stored in persistent storage and are retrieved from database.

7. EJB – MESSAGE DRIVEN BEANS

A message driven bean is a type of enterprise bean, which is invoked by EJB container when it receives a message from queue or topic. Message driven bean is a stateless bean and is used to do task asynchronously.

To demonstrate use of message driven bean, we will make use of EJB-persistence chapter and we need to do the following tasks:

- Step 1. Create table in database (Refer to *EJB-Persistence* chapter).
- Step 2. Create Entity class corresponding to table (Refer to *EJB-Persistence* chapter).
- Step 3. Create DataSource and Persistence Unit (Refer to *EJB-Persistence* chapter).
- Step 4. Create a stateless EJB having EntityManager instance (Refer to *EJB-Persistence* chapter).
- Step 5. Update stateless ejb.Add methods to add records and get records from database via entity manager (Refer to *EJB-Persistence* chapter).
- Step 6. Create a Queue named **BookQueue** in JBoss **default** application directory.
- Step 7. A console based application client will send message to this queue.
- Step 8. Create a Message driven bean, which will use the stateless bean to persist the client data.
- Step 9. EJB Container of jboss will call the above message driven bean and pass it the message that client will be sending to.

Create Queue

Create a file named jbossmq-destinations-service.xml if not exists in **<JBoss Installation Folder> > server > default > deploy** folder.

Here we are creating a queue named BookQueue:

jbossmq-destinations-service.xml

```
<mbean code="org.jboss.mq.server.jmx.Queue"
   name="jboss.mq.destination:service=Queue,name=BookQueue">
   <depends optional-attribute-name="DestinationManager">
      jboss.mq:service=DestinationManager
   </depends>
</mbean>
```

When you start the JBoss, you will see a similar entry in jboss log.

```
...
10:37:06,167 INFO  [QueueService] Queue[/queue/BookQueue] started,
fullSize=200000, pageSize=2000, downCacheSize=2000
...
```

Create Message Driven Bean

```
@MessageDriven(
   name = "BookMessageHandler",
   activationConfig = {
      @ActivationConfigProperty( propertyName = "destinationType",
                                 propertyValue = "javax.jms.Queue"),
      @ActivationConfigProperty( propertyName = "destination",
                                 propertyValue ="/queue/BookQueue")
   }
)
public class LibraryMessageBean implements MessageListener {

   @Resource
   private MessageDrivenContext mdctx;

   @EJB
   LibraryPersistentBeanRemote libraryBean;

   public LibraryMessageBean(){
   }

   public void onMessage(Message message) {
   }
}
```

- LibraryMessageBean is annotated with @MessageDriven annotation to mark it as message driven bean.
- Its properties are defined as destinationType - Queue and destination - /queue/BookQueue.
- It implements MessageListener interface, which exposes onMessage method.

- It has MessgeDrivenContext as a resource.
- LibraryPersistentBeanRemote stateless bean is injected in this bean for persistence purpose.

Build the EjbComponent project and deploy it on JBoss. After building and deploying the EJB module, we need a client to send a message to jboss queue.

Example Application

Let us create a test EJB application to test Message Driven Bean.

Step	Description
1	Create a project with a name *EjbComponent* under a package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. You can also use the project created in *EJB - Create Application* chapter as such for this chapter to understand EJB persistence concepts.
2	Create *Book.java* under package *com.tutorialspoint.entity* as created in *EJB-Persistence* chapter.
3	Create *LibraryPersistentBean.java* and *LibraryPersistentBeanRemote* as created in *EJB-Persistence* chapter.
4	Create *jboss-ds.xml* in **EjbComponent > setup** folder and *persistence.xml* in **EjbComponent > src > conf** folder. These folders can be seen in files tab in Netbeans as created in *EJB-Persistence* chapter.
5	Create *LibraryMessageBean.java* under a package *com.tutorialspoint.messagebean* and modify it as shown below.
6	Create *BookQueue* queue in Jboss as described above.
7	Clean and Build the application to make sure business logic is working as per the requirements.
8	Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.
9	Now create the EJB client, a console based application in the same way as explained in the *EJB - Create Application* chapter under topic **Create Client to access EJB**. Modify it as shown below.

EJBComponent (EJB Module)

LibraryMessageBean.java

```
package com.tuturialspoint.messagebean;


import com.tutorialspoint.entity.Book;

import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;

import javax.annotation.Resource;
```

```
import javax.ejb.ActivationConfigProperty;
import javax.ejb.EJB;
import javax.ejb.MessageDriven;
import javax.ejb.MessageDrivenContext;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.ObjectMessage;

@MessageDriven(
   name = "BookMessageHandler",
   activationConfig = {
      @ActivationConfigProperty( propertyName = "destinationType",
                                 propertyValue = "javax.jms.Queue"),
      @ActivationConfigProperty( propertyName = "destination",
                                 propertyValue ="/queue/BookQueue")
   }
)
public class LibraryMessageBean implements MessageListener {

   @Resource
   private MessageDrivenContext mdctx;

   @EJB
   LibraryPersistentBeanRemote libraryBean;

   public LibraryMessageBean(){
   }

   public void onMessage(Message message) {
      ObjectMessage objectMessage = null;
      try {
         objectMessage = (ObjectMessage) message;
         Book book = (Book) objectMessage.getObject();
         libraryBean.addBook(book);
```

```
      } catch (JMSException ex) {
         mdctx.setRollbackOnly();
      }
   }
}
```

EJBTester (EJB Client)

EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.entity.Book;
import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.jms.ObjectMessage;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueSender;
import javax.jms.QueueSession;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
```

```
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testMessageBeanEjb();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options \n1. Add Book\n2. Exit \nEnter Choice: ");
   }

   private void testMessageBeanEjb(){

      try {
         int choice = 1;
         Queue queue = (Queue) ctx.lookup("/queue/BookQueue");
         QueueConnectionFactory factory =
```

```
(QueueConnectionFactory) ctx.lookup("ConnectionFactory");
QueueConnection connection =  factory.createQueueConnection();
QueueSession session =
connection.createQueueSession(false, QueueSession.AUTO_ACKNOWLEDGE);
QueueSender sender = session.createSender(queue);

while (choice != 2) {
   String bookName;
   showGUI();
   String strChoice = brConsoleReader.readLine();
   choice = Integer.parseInt(strChoice);
   if (choice == 1) {
      System.out.print("Enter book name: ");
      bookName = brConsoleReader.readLine();
      Book book = new Book();
      book.setName(bookName);
      ObjectMessage objectMessage =
         session.createObjectMessage(book);
      sender.send(objectMessage);
   } else if (choice == 2) {
      break;
   }
}

LibraryPersistentBeanRemote libraryBean =
(LibraryPersistentBeanRemote)
ctx.lookup("LibraryPersistentBean/remote");

List<Book> booksList = libraryBean.getBooks();

System.out.println("Book(s) entered so far: " + booksList.size());
int i = 0;
for (Book book:booksList) {
   System.out.println((i+1)+". " + book.getName());
   i++;
```

```
         }
      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
   }
}
```

EJBTester performs the following tasks:

- Load properties from jndi.properties and initialize the InitialContext object.
- In testStatefulEjb() method, jndi lookup is done with the name - "/queue/BookQueue" to obtain reference of queue available in Jboss. Then sender is created using queue session.
- Then user is shown a library store User Interface and he/she is asked to enter choice.
- If user enters 1, the system asks for book name and sender sends the book name to queue. When JBoss container receives this message in queue, it calls our message driven bean's onMessage method. Our message driven bean then saves book using stateful session bean addBook() method. Session Bean is persisting the book in database via EntityManager call.
- If user enters 2, then another jndi lookup is done with name - "LibraryStatefulSessionBean/remote" to obtain the remote business object (stateful EJB) again and listing of books is done.

Run Client to Access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console:

```
run:
```

```
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
```

```
Enter Choice: 1
```

```
Enter book name: Learn EJB
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 2
1. learn java
1. learn EJB
BUILD SUCCESSFUL (total time: 15 seconds)
```

The output shown above states that our Message driven bean is receiving the message and storing the book in persistent storage and books are retrieved from the database.

8. EJB – ANNOTATIONS

Annotations were introduced in Java 5.0. The purpose of having annotations is to attach additional information in the class or a meta-data of a class within its source code. In EJB 3.0, annotations are used to describe configuration meta-data in EJB classes. By this way, EJB 3.0 eliminates the need to describe configuration data in configuration XML files.

EJB container uses compiler tool to generate required artifacts like interfaces, deployment descriptors by reading those annotations. Following is the list of commonly used annotations.

Sr. No.	Name	Description
1	javax.ejb.Stateless	Specifies that a given EJB class is a stateless session bean. **Attributes** **name** - Used to specify name of the session bean. **mappedName** - Used to specify the JNDI name of the session bean. **description** - Used to provide description of the session bean.
2	javax.ejb.Stateful	Specifies that a given EJB class is a stateful session bean. **Attributes** **name** - Used to specify name of the session bean. **mappedName** - Used to specify the JNDI name of the session bean. **description** - Used to provide description of the session bean.
3	javax.ejb.MessageDrivenBean	Specifies that a given EJB class is a message driven bean. **Attributes** **name** - Used to specify name of the message driven bean. **messageListenerInterface** - Used to specify message listener interface for the message driven bean. **activationConfig** - Used to specify the configuration details of the message-driven bean in an operational environment of the message driven bean. **mappedName** - Used to specify the JNDI name of the session bean. **description** - Used to provide description of the session bean.

4	javax.ejb.EJB	Used to specify or inject a dependency as EJB instance into another EJB. **Attributes** **name** - Used to specify name, which will be used to locate the referenced bean in the environment. **beanInterface** - Used to specify the interface type of the referenced bean. **beanName** - Used to provide name of the referenced bean. **mappedName** - Used to specify the JNDI name of the referenced bean. **description** - Used to provide description of the referenced bean.
5	javax.ejb.Local	Used to specify Local interface(s) of a session bean. This local interface states the business methods of the session bean (which can be stateless or stateful). This interface is used to expose the business methods to local clients, which are running in the same deployment/application as EJB. **Attributes** **value** - Used to specify the list of local interfaces as an array of interfaces.
6	javax.ejb.Remote	Used to specify Remote interface(s) of a session bean. This remote interface states the business methods of the session bean (which can be stateless or stateful). This interface is used to expose the business methods to remote clients, which are running in different deployment/application as EJB. **Attributes** **value** - Used to specify the list of remote interfaces as an array of interfaces.
7	javax.ejb.ActivationConfigProperty	Used to specify properties required for a message driven bean. For example, end point, destination, message selector etc. This annotation is passed as a parameter to activationConfig attribute of javax.ejb.MessageDrivenBean annotation. **Attributes** **propertyName** - name of the property. **propertyValue** - value of the property.

8	javax.ejb.PostActivate	Used to specify callback method of EJB lifecycle. This method will be called when EJB container just activated/reactivated the bean instance. This interface is used to expose the business methods to local clients, which are running in same deployment/application as EJB.

9. EJB – CALLBACKS

Callback is a mechanism by which the life cycle of an enterprise bean can be intercepted. EJB 3.0 specification has specified callbacks for which callback handler methods are created. EJB Container calls these callbacks. We can define callback methods in the EJB class itself or in a separate class. EJB 3.0 has provided many annotations for callbacks.

Following is the list of callback annotations for stateless bean:

Annotation	Description
@PostConstruct	Invoked when a bean is created for the first time.
@PreDestroy	Invoked when a bean is removed from the bean pool or is destroyed.

Following is the list of callback annotations for stateful bean.

Annotation	Description
@PostConstruct	Invoked when a bean is created for the first time.
@PreDestroy	Invoked when a bean is removed from the bean pool or is destroyed.
@PostActivate	Invoked when a bean is loaded to be used.
@PrePassivate	Invoked when a bean is put back to bean pool.

Following is the list of callback annotations for message driven bean:

Annotation	Description
@PostConstruct	Invoked when a bean is created for the first time.
@PreDestroy	Invoked when a bean is removed from the bean pool or is destroyed.

Following is the list of callback annotations for entity bean:

Annotation	Description
@PrePersist	Invoked when an entity is created in database.
@PostPersist	Invoked after an entity is created in database.
@PreRemove	Invoked when an entity is deleted from the database.
@PostRemove	Invoked after an entity is deleted from the database.
@PreUpdate	Invoked before an entity is to be updated in the database.
@PostLoad	Invoked when a record is fetched from database and loaded into the entity.

Example Application

Let us create a test EJB application to test various callbacks in EJB.

Step	Description
1	Create a project with a name *EjbComponent* under a package *com.tutorialspoint.stateless* as explained in the *EJB - Create Application* chapter. You can also use the project created in *EJB - Persistence* chapter as such for this chapter to add various callbacks to EJB.
2	Create *LibrarySessionBean.java* and *LibrarySessionBeanRemote* as explained in the *EJB - Create Application* chapter. Keep rest of the files unchanged.
3	Use Beans created in the *EJB - Persistence* chapter. Add callback methods as shown below. Keep rest of the files unchanged.
4	Create a java class *BookCallbackListener* under package *com.tutorialspoint.callback*. This class will demonstrates the separation of callback methods.
5	Clean and Build the application to make sure business logic is working as per the requirements.
6	Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.
7	Now create the EJB client, a console based application in the same way as explained in the *EJB - Create Application* chapter under topic **Create Client to access EJB**.

EJBComponent (EJB Module)

BookCallbackListener.java

```
package com.tutorialspoint.callback;


import javax.persistence.PrePersist;
import javax.persistence.PostLoad;
import javax.persistence.PostPersist;
import javax.persistence.PostRemove;
import javax.persistence.PostUpdate;
import javax.persistence.PreRemove;
import javax.persistence.PreUpdate;
import com.tutorialspoint.entity.Book;


public class BookCallbackListener {
```

```
@PrePersist
public void prePersist(Book book){
   System.out.println("BookCallbackListener.prePersist:"
      + "Book to be created with book id: "+book.getId());
}

@PostPersist
public void postPersist(Object book){
   System.out.println("BookCallbackListener.postPersist::"
      + "Book created with book id: "+((Book)book).getId());
}

@PreRemove
public void preRemove(Book book)
{
   System.out.println("BookCallbackListener.preRemove:"
      + " About to delete Book: " + book.getId());
}

@PostRemove
public void postRemove(Book book)
{
   System.out.println("BookCallbackListener.postRemove::"
      + " Deleted Book: " + book.getId());
}

@PreUpdate
public void preUpdate(Book book)
{
   System.out.println("BookCallbackListener.preUpdate::"
      + " About to update Book: " + book.getId());
}

@PostUpdate
public void postUpdate(Book book)
```

```
   {
      System.out.println("BookCallbackListener.postUpdate::"
         + " Updated Book: " + book.getId());
   }

   @PostLoad
   public void postLoad(Book book)
   {
      System.out.println("BookCallbackListener.postLoad::"
         + " Loaded Book: " + book.getId());
   }
}
```

Book.java

```
package com.tutorialspoint.entity;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EntityListeners;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="books")
public class Book implements Serializable{

   private int id;
   private String name;

   public Book(){
   }
```

```
   @Id
   @GeneratedValue(strategy= GenerationType.IDENTITY)
   @Column(name="id")
   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
}
```

LibraryStatefulSessionBean.java

```
package com.tutorialspoint.stateful;

import java.util.ArrayList;
import java.util.List;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.ejb.PostActivate;
import javax.ejb.PrePassivate;
import javax.ejb.Stateful;

@Stateful
public class LibraryStatefulSessionBean
   implements LibraryStatefulSessionBeanRemote {
   List<String> bookShelf;
```

```
public LibraryStatefulSessionBean(){
   bookShelf = new ArrayList<String>();
}

public void addBook(String bookName) {
   bookShelf.add(bookName);
}

public List<String> getBooks() {
   return bookShelf;
}

@PostConstruct
public void postConstruct(){
   System.out.println("LibraryStatefulSessionBean.postConstruct::"
      + " bean created.");
}

@PreDestroy
public void preDestroy(){
   System.out.println("LibraryStatefulSessionBean.preDestroy:"
      + " bean removed.");
}

@PostActivate
public void postActivate(){
   System.out.println("LibraryStatefulSessionBean.postActivate:"
      + " bean activated.");
}

@PrePassivate
public void prePassivate(){
   System.out.println("LibraryStatefulSessionBean.prePassivate:"
      + " bean passivated.");
```

```
   }
}
```

LibraryStatefulSessionBeanRempote.java

```
package com.tutorialspoint.stateful;

import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryStatefulSessionBeanRemote {
   void addBook(String bookName);
   List getBooks();
}
```

LibraryPersistentBean.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
public class LibraryPersistentBean
   implements LibraryPersistentBeanRemote {

   public LibraryPersistentBean(){}

   @PersistenceContext(unitName="EntityEjbPU")
   private EntityManager entityManager;
```

```
   public void addBook(Book book) {
      entityManager.persist(book);
   }

   public List<Book> getBooks() {
      return entityManager.createQuery("From Book")
         .getResultList();
   }

   @PostConstruct
   public void postConstruct(){
      System.out.println("postConstruct:: LibraryPersistentBean session bean"
         + " created with entity Manager object: ");
   }

   @PreDestroy
   public void preDestroy(){
      System.out.println("preDestroy: LibraryPersistentBean session"
      + " bean is removed ");
   }
}
```

LibraryPersistentBeanRemote.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryPersistentBeanRemote {

   void addBook(Book bookName);

   List<Book> getBooks();
```

```
}
```

- As soon as you deploy the EjbComponent project on JBOSS, notice the jboss log.
- JBoss has automatically created a JNDI entry for our session bean - **LibraryPersistentBean/remote**.
- We will be using this lookup string to get remote business object of type - **com.tutorialspoint.stateless.LibraryPersistentBeanRemote**

JBoss Application Server Log Output

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:

   LibraryPersistentBean/remote - EJB3.x Default Remote Business Interface

   LibraryPersistentBean/remote-
com.tutorialspoint.stateless.LibraryPersistentBeanRemote - EJB3.x Remote
Business Interface

16:30:02,723 INFO  [SessionSpecContainer] Starting
jboss.j2ee:jar=EjbComponent.jar,name=LibraryPersistentBean,service=EJB3

16:30:02,723 INFO  [EJBContainer] STARTED EJB:
com.tutorialspoint.stateless.LibrarySessionBeanRemote ejbName:
LibraryPersistentBean

...
```

EJBTester (EJB Client)

jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory

java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces

java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service.
- InitialContext object will be used to lookup stateless session bean.

EJBTester.java

```
package com.tutorialspoint.test;
```

```
import com.tutorialspoint.stateful.LibrarySessionBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();
```

```
      ejbTester.testEntityEjb();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options \n1. Add Book\n2. Exit \nEnter Choice: ");
   }

   private void testEntityEjb(){
      try {
      int choice = 1;

      LibraryPersistentBeanRemote libraryBean =
      (LibraryPersistentBeanRemote)
      ctx.lookup("LibraryPersistentBean/remote");

      while (choice != 2) {
         String bookName;
         showGUI();
         String strChoice = brConsoleReader.readLine();
         choice = Integer.parseInt(strChoice);
         if (choice == 1) {
            System.out.print("Enter book name: ");
            bookName = brConsoleReader.readLine();
            Book book = new Book();
            book.setName(bookName);
            libraryBean.addBook(book);
         } else if (choice == 2) {
            break;
         }
      }

      List<Book> booksList = libraryBean.getBooks();
```

```
      System.out.println("Book(s) entered so far: " + booksList.size());
      int i = 0;
      for (Book book:booksList) {
         System.out.println((i+1)+". " + book.getName());
         i++;
      }

      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
   }
}
```

EJBTester performs the following tasks:

- Load properties from jndi.properties and initialize the InitialContext object.
- In testStatelessEjb() method, jndi lookup is done with the name - "LibrarySessionBean/remote" to obtain the remote business object (stateless EJB).
- Then user is shown a library store User Interface and he/she is asked to enter a choice.
- If the user enters 1, the system asks for book name and saves the book using stateless session bean addBook() method. Session Bean is storing the book in the database.
- If user enters 2, system retrieves books using stateless session bean getBooks() method and exits.

Run Client to Access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 1
Enter book name: Learn Java
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 1
1. Learn Java
BUILD SUCCESSFUL (total time: 13 seconds)
```

JBoss Application Server Log Output

You can find the following callback entries in JBoss log:

```
14:08:34,293 INFO  [STDOUT] postConstruct:: LibraryPersistentBean session bean
created with entity Manager object
...
16:39:09,484 INFO  [STDOUT] BookCallbackListener.prePersist:: Book to be
created with book id: 0
16:39:09,531 INFO  [STDOUT] BookCallbackListener.postPersist:: Book created
with book id: 1
16:39:09,900 INFO  [STDOUT] BookCallbackListener.postLoad:: Loaded Book: 1
...
```

10. EJB – TIMER SERVICE

Timer Service is a mechanism by which scheduled application can be build. For example, salary slip generation on the 1st of every month. EJB 3.0 specification has specified @Timeout annotation, which helps in programming the EJB service in a stateless or message driven bean. EJB Container calls the method, which is annotated by @Timeout.

EJB Timer Service is a service provided by EJB container, which helps to create timer and to schedule callback when timer expires.

Steps to Create Timer

Inject SessionContext in bean using @Resource annotation:

```
@Stateless
public class TimerSessionBean {

   @Resource
   private SessionContext context;
   ...
}
```

Use SessionContext object to get TimerService and to create timer. Pass time in milliseconds and message.

```
public void createTimer(long duration) {
   context.getTimerService().createTimer(duration, "Hello World!");
}
```

Steps to Use Timer

Use @Timeout annotation to a method. Return type should be void and pass a parameter of type Timer. We are canceling the timer after first execution otherwise it will keep running after fix intervals.

```
@Timeout
public void timeOutHandler(Timer timer){
   System.out.println("timeoutHandler : " + timer.getInfo());
   timer.cancel();
}
```

Example Application

Let us create a test EJB application to test Timer Service in EJB.

Step	Description
1	Create a project with a name *EjbComponent* under a package *com.tutorialspoint.timer* as explained in the *EJB - Create Application* chapter.
2	Create *TimerSessionBean.java* and *TimerSessionBeanRemote* as explained in the *EJB - Create Application* chapter. Keep rest of the files unchanged.
3	Clean and Build the application to make sure business logic is working as per the requirements.
4	Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.
5	Now create the EJB client, a console based application in the same way as explained in the *EJB - Create Application* chapter under topic **Create Client to access EJB**.

EJBComponent (EJB Module)

TimerSessionBean.java

```
package com.tutorialspoint.timer;

import javax.annotation.Resource;
import javax.ejb.SessionContext;
import javax.ejb.Timer;
import javax.ejb.Stateless;
import javax.ejb.Timeout;

@Stateless
public class TimerSessionBean implements TimerSessionBeanRemote {

   @Resource
   private SessionContext context;

   public void createTimer(long duration) {
      context.getTimerService().createTimer(duration, "Hello World!");
   }

```

```
   @Timeout
   public void timeOutHandler(Timer timer){
      System.out.println("timeoutHandler : " + timer.getInfo());
      timer.cancel();
   }
}
```

TimerSessionBeanRemote.java

```
package com.tutorialspoint.timer;

import javax.ejb.Remote;

@Remote
public interface TimerSessionBeanRemote {
   public void createTimer(long milliseconds);
}
```

- As soon as you deploy the EjbComponent project on JBOSS, notice the jboss log.
- JBoss has automatically created a JNDI entry for our session bean - **TimerSessionBean/remote**.
- We will using this lookup string to get remote business object of type - **com.tutorialspoint.timer.TimerSessionBeanRemote**

JBoss Application Server Log Output

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:
   TimerSessionBean/remote - EJB3.x Default Remote Business Interface
   TimerSessionBean/remote-com.tutorialspoint.timer.TimerSessionBeanRemote -
EJB3.x Remote Business Interface
16:30:02,723 INFO  [SessionSpecContainer] Starting
jboss.j2ee:jar=EjbComponent.jar,name=TimerSessionBean,service=EJB3
16:30:02,723 INFO  [EJBContainer] STARTED EJB:
com.tutorialspoint.timer.TimerSessionBeanRemote ejbName: TimerSessionBean
...
```

EJBTester (EJB Client)

jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service.
- InitialContext object will be used to lookup stateless session bean.

EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateful.TimerSessionBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
```

```
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testTimerService();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options \n1. Add Book\n2. Exit \nEnter Choice: ");
   }

   private void testTimerService(){
      try {
         TimerSessionBeanRemote timerServiceBean =
(TimerSessionBeanRemote)ctx.lookup("TimerSessionBean/remote");

         System.out.println("["+(new Date()).toString()+ "]" + "timer
created.");
         timerServiceBean.createTimer(2000);

      } catch (NamingException ex) {
         ex.printStackTrace();
      }
   }
```

```
}
```

EJBTester is doing the following tasks.

- Load properties from jndi.properties and initialize the InitialContext object.
- In testTimerService() method, jndi lookup is done with the name - "TimerSessionBean/remote" to obtain the remote business object (timer stateless EJB).
- Then createTimer is invoked passing 2000 milliseconds as schedule time.
- EJB Container calls the timeoutHandler method after 2 seconds.

Run Client to Access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
[Wed Jun 19 11:35:47 IST 2013]timer created.
BUILD SUCCESSFUL (total time: 0 seconds)
```

JBoss Application Server Log Output

```
You can find the following callback entries in JBoss log
...
11:35:49,555 INFO  [STDOUT] timeoutHandler : Hello World!
...
```

11. EJB – DEPENDENCY INJECTION

EJB 3.0 specification provides annotations, which can be applied on fields or setter methods to inject dependencies. EJB Container uses the global JNDI registry to locate the dependency. Following annotations are used in EJB 3.0 for dependency injection.

- **@EJB** - used to inject other EJB reference.
- **@Resource** - used to inject datasource or singleton services like sessionContext, timerService etc.

Steps to Use @EJB

@EJB can be used on fields or on methods in the following manner:

```
public class LibraryMessageBean implements MessageListener {
   //dependency injection on field.
   @EJB
   LibraryPersistentBeanRemote libraryBean;
   ...
}
public class LibraryMessageBean implements MessageListener {

   LibraryPersistentBeanRemote libraryBean;

   //dependency injection on method.
   @EJB(beanName="com.tutorialspoint.stateless.LibraryPersistentBean")
   public void setLibraryPersistentBean(
   LibraryPersistentBeanRemote libraryBean)
   {
      this.libraryBean = libraryBean;
   }
   ...
}
```

Steps to use @Resource

@Resource is normally used to inject EJB Container provided singletons.

```
public class LibraryMessageBean implements MessageListener {
   @Resource
   private MessageDrivenContext mdctx;
   ...
}
```

Example Application

Let us create a test EJB application to test Dependency Injection Service in EJB.

Step	Description
1	Create a project with a name *EjbComponent* under a package *com.tutorialspoint.timer* as explained in the *EJB - Create Application* chapter.
3	Use Beans created in the *EJB - Message Driven Bean* chapter. Keep rest of the files unchanged.
5	Clean and Build the application to make sure business logic is working as per the requirements.
6	Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.
7	Now create the EJB client, a console based application in the same way as explained in the *EJB - Create Application* chapter under topic **Create Client to access EJB**.

EJBComponent (EJB Module)

LibraryMessageBean.java

```
package com.tuturialspoint.messagebean;


import com.tutorialspoint.entity.Book;
import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;
import javax.annotation.Resource;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.EJB;
import javax.ejb.MessageDriven;
import javax.ejb.MessageDrivenContext;
```

```
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.ObjectMessage;

@MessageDriven(
   name = "BookMessageHandler",
   activationConfig = {
      @ActivationConfigProperty( propertyName = "destinationType",
                                 propertyValue = "javax.jms.Queue"),
      @ActivationConfigProperty( propertyName = "destination",
                                 propertyValue ="/queue/BookQueue")
   }
)
public class LibraryMessageBean implements MessageListener {

   @Resource
   private MessageDrivenContext mdctx;

   @EJB
   LibraryPersistentBeanRemote libraryBean;

   public LibraryMessageBean(){
   }

   public void onMessage(Message message) {
      ObjectMessage objectMessage = null;
      try {
         objectMessage = (ObjectMessage) message;
         Book book = (Book) objectMessage.getObject();
         libraryBean.addBook(book);

      } catch (JMSException ex) {
         mdctx.setRollbackOnly();
      }
   }
```

```
}
```

EJBTester (EJB Client)

EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.entity.Book;
import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.jms.ObjectMessage;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueSender;
import javax.jms.QueueSession;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
```

```
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testMessageBeanEjb();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options \n1. Add Book\n2. Exit \nEnter Choice: ");
   }

   private void testMessageBeanEjb(){

      try {
         int choice = 1;
         Queue queue = (Queue) ctx.lookup("/queue/BookQueue");
         QueueConnectionFactory factory =
         (QueueConnectionFactory) ctx.lookup("ConnectionFactory");
         QueueConnection connection =  factory.createQueueConnection();
         QueueSession session = connection.createQueueSession(
         false, QueueSession.AUTO_ACKNOWLEDGE);
         QueueSender sender = session.createSender(queue);
```

```
        while (choice != 2) {
            String bookName;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               Book book = new Book();
               book.setName(bookName);
               ObjectMessage objectMessage =
               session.createObjectMessage(book);
               sender.send(objectMessage);
            } else if (choice == 2) {
               break;
            }
        }

        LibraryPersistentBeanRemote libraryBean =
        (LibraryPersistentBeanRemote)
            ctx.lookup("LibraryPersistentBean/remote");

        List<Book> booksList = libraryBean.getBooks();

        System.out.println("Book(s) entered so far: "
        + booksList.size());
        int i = 0;
        for (Book book:booksList) {
            System.out.println((i+1)+". " + book.getName());
            i++;
        }
     } catch (Exception e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
```

```
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
   }
}
```

EJBTester performs the following tasks:

- Load properties from jndi.properties and initialize the InitialContext object.
- In testStatefulEjb() method, jndi lookup is done with the name - "/queue/BookQueue" to obtain reference of queue available in Jboss. Then sender is created using queue session.
- Then the user is shown a library store User Interface and he/she is asked to enter a choice.
- If the user enters 1, the system asks for book name and sender sends the book name to queue. When JBoss container receives this message in queue, it calls our message driven bean's onMessage method. Our message driven bean then saves book using stateful session bean addBook() method. Session Bean is persisting the book in database via EntityManager call.
- If the user enters 2, then another jndi lookup is done with the name - "LibraryStatefulSessionBean/remote" to obtain the remote business object (stateful EJB) again and listing of books is done.

Run Client to Access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
**********************
Welcome to Book Store
**********************
Options
```

```
1. Add Book
2. Exit
Enter Choice: 1
Enter book name: Learn EJB

**********************

Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 2
1. learn java
1. learn EJB
BUILD SUCCESSFUL (total time: 15 seconds)
```

The output shown above states that our Message driven bean is receiving the message and storing book in persistent storage and books are retrieved from database.

Our message driven bean is using LibraryPersistentBean injected into it using @EJB annotation and in case of exception, MessageDrivenContext object is used to roll-back the transaction.

12. EJB – INTERCEPTORS

EJB 3.0 provides specification to intercept business methods calls using methods annotated with @AroundInvoke annotation. An interceptor method is called by ejbContainer before business method call it is intercepting. Following is the example signature of an interceptor method

```
@AroundInvoke
public Object methodInterceptor(InvocationContext ctx) throws Exception
{
   System.out.println("*** Intercepting call to LibraryBean method: "
   + ctx.getMethod().getName());
   return ctx.proceed();
}
```

Interceptor methods can be applied or bound at three levels

- **Default** - Default interceptor is invoked for every bean within deployment. Default interceptor can be applied only via xml (ejb-jar.xml).
- **Class** - Class level interceptor is invoked for every method of the bean. Class level interceptor can be applied both by annotation of via xml(ejb-jar.xml).
- **Method** - Method level interceptor is invoked for a particular method of the bean. Method level interceptor can be applied both by annotation of via xml(ejb-jar.xml).

We are discussing Class level interceptor here.

Interceptor Class

```
package com.tutorialspoint.interceptor;

import javax.interceptor.AroundInvoke;
import javax.interceptor.InvocationContext;

public class BusinessInterceptor {
   @AroundInvoke
   public Object methodInterceptor(InvocationContext ctx) throws Exception
   {
      System.out.println("*** Intercepting call to LibraryBean method: "
      + ctx.getMethod().getName());
```

```
      return ctx.proceed();
   }
}
```

Remote Interface

```
import javax.ejb.Remote;

@Remote
public interface LibraryBeanRemote {
   //add business method declarations
}
```

Intercepted Stateless EJB

```
@Interceptors ({BusinessInterceptor.class})
@Stateless
public class LibraryBean implements LibraryBeanRemote {
   //implement business method
}
```

Example Application

Let us create a test EJB application to test intercepted stateless EJB.

Step	Description
1	Create a project with a name *EjbComponent* under a package *com.tutorialspoint.interceptor* as explained in the *EJB - Create Application* chapter. You can also use the project created in *EJB - Create Application* chapter as such for this chapter to understand intercepted EJB concepts.
2	Create *LibraryBean.java* and *LibraryBeanRemote* under package *com.tutorialspoint.interceptor* as explained in the *EJB - Create Application* chapter. Keep rest of the files unchanged.
3	Clean and Build the application to make sure business logic is working as per the requirements.
4	Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.
5	Now create the ejb client, a console based application in the same way as explained in the *EJB - Create Application* chapter under topic **Create Client to access EJB**.

EJBComponent (EJB Module)

LibraryBeanRemote.java

```
package com.tutorialspoint.interceptor;

import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryBeanRemote {
   void addBook(String bookName);
   List getBooks();
}
```

LibraryBean.java

```
package com.tutorialspoint.interceptor;

import java.util.ArrayList;
import java.util.List;
import javax.ejb.Stateless;
import javax.interceptor.Interceptors;

@Interceptors ({BusinessInterceptor.class})
@Stateless
public class LibraryBean implements LibraryBeanRemote {

   List<String> bookShelf;

   public LibraryBean(){
      bookShelf = new ArrayList<String>();
   }

   public void addBook(String bookName) {
      bookShelf.add(bookName);
   }
```

```
   public List<String> getBooks() {
      return bookShelf;
   }
}
```

- As soon as you deploy the EjbComponent project on JBOSS, notice the jboss log.
- JBoss has automatically created a JNDI entry for our session bean - **LibraryBean/remote**.
- We will using this lookup string to get remote business object of type - **com.tutorialspoint.interceptor.LibraryBeanRemote**

JBoss Application Server Log Output

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:
   LibraryBean/remote - EJB3.x Default Remote Business Interface
   LibraryBean/remote-com.tutorialspoint.interceptor.LibraryBeanRemote - EJB3.x
Remote Business Interface
16:30:02,723 INFO  [SessionSpecContainer] Starting
jboss.j2ee:jar=EjbComponent.jar,name=LibraryBean,service=EJB3
16:30:02,723 INFO  [EJBContainer] STARTED EJB:
com.tutorialspoint.interceptor.LibraryBeanRemote ejbName: LibraryBean
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:

   LibraryBean/remote - EJB3.x Default Remote Business Interface
   LibraryBean/remote-com.tutorialspoint.interceptor.LibraryBeanRemote - EJB3.x
Remote Business Interface
...
```

EJBTester (EJB Client)

jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service.
- InitialContext object will be used to lookup stateless session bean.

EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateful.LibraryBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
```

```
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testInterceptedEjb();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options \n1. Add Book\n2. Exit \nEnter Choice: ");
   }

   private void testInterceptedEjb(){

      try {
         int choice = 1;

         LibraryBeanRemote libraryBean =
         LibraryBeanRemote)ctx.lookup("LibraryBean/remote");

         while (choice != 2) {
            String bookName;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               Book book = new Book();
               book.setName(bookName);
```

```
               libraryBean.addBook(book);
            } else if (choice == 2) {
               break;
            }
         }

         List<Book> booksList = libraryBean.getBooks();

         System.out.println("Book(s) entered so far: " + booksList.size());
         int i = 0;
         for (Book book:booksList) {
            System.out.println((i+1)+". " + book.getName());
            i++;
         }
      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
   }
}
```

EJBTester performs the following tasks:

- Load properties from jndi.properties and initialize the InitialContext object.
- In testInterceptedEjb() method, jndi lookup is done with the name - "LibraryBean/remote" to obtain the remote business object (stateless EJB).
- Then user is shown a library store User Interface and he/she is asked to enter a choice.

- If the user enters 1, the system asks for book name and saves the book using stateless session bean addBook() method. Session Bean is storing the book in its instance variable.
- If the user enters 2, the system retrieves books using stateless session bean getBooks() method and exits.

Run Client to Access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 1
Enter book name: Learn Java
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 1
1. Learn Java
BUILD SUCCESSFUL (total time: 13 seconds)
```

JBoss Application Server Log Output

Verify the following output in JBoss Application server log output.

```
....
09:55:40,741 INFO  [STDOUT] *** Intercepting call to LibraryBean method:
addBook
```

```
09:55:43,661 INFO  [STDOUT] *** Intercepting call to LibraryBean method:
getBooks
```

13. EJB – EMBEDDABLE OBJECTS

EJB 3.0 provides option to embed JAVA POJO (Plain Old Java Object) into an entity bean and allows to map column names with the methods of the embedded POJO class. A java POJO to be embedded must be annotated as @Embeddable.

```
@Embeddable
public class Publisher implements Serializable{
   private String name;
   private String address;
   ...
}
```

The above class can be embedded using @Embedded annotation

```
@Entity
public class Book implements Serializable{
   private int id;
   private String name;
   private Publisher publisher;
   ...
   @Embedded
   @AttributeOverrides({
   @AttributeOverride(name = "name",
                      column = @Column(name = "PUBLISHER")),
   @AttributeOverride(name = "address",
                      column = @Column(name = "PUBLISHER_ADDRESS"))
   })
   public Publisher getPublisher() {
      return publisher;
   }
   ...
}
```

Example Application

Let us create a test EJB application to test embedded objects in EJB 3.0.

Step	Description
1	Create a project with a name *EjbComponent* under a package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. Please use the project created in *EJB - Persistence* chapter as such for this chapter to understand embedded objects in EJB concepts.
2	Create *Publisher.java* under package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. Keep rest of the files unchanged.
3	Create *Book.java* under package *com.tutorialspoint.entity*. Use *EJB - Persistence* chapter as reference. Keep rest of the files unchanged.
4	Clean and Build the application to make sure business logic is working as per the requirements.
5	Finally, deploy the application in the form of a jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.
6	Now create the EJB client, a console based application in the same way as explained in the *EJB - Create Application* chapter under topic **Create Client to access EJB**.

Create/Alter Book Table

```
CREATE TABLE book (
   id     integer PRIMARY KEY,
   name   varchar(50)
);
Alter table book add publisher varchar(100);
Alter table book add publisher_address varchar(200);
```

EJBComponent (EJB Module)

Publisher.java

```
package com.tutorialspoint.entity;

import java.io.Serializable;
import javax.persistence.Embeddable;

@Embeddable
```

```
public class Publisher implements Serializable{

   private String name;
   private String address;

   public Publisher(){}

   public Publisher(String name, String address){
      this.name = name;
      this.address = address;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getAddress() {
      return address;
   }

   public void setAddress(String address) {
      this.address = address;
   }

   public String toString(){
      return name + "," + address;
   }
}
```

Book.java

```
package com.tutorialspoint.entity;

import com.tutorialspoint.callback.BookCallbackListener;
import java.io.Serializable;
import javax.persistence.AttributeOverride;
import javax.persistence.AttributeOverrides;
import javax.persistence.Column;
import javax.persistence.Embedded;
import javax.persistence.Entity;
import javax.persistence.EntityListeners;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="book")
public class Book implements Serializable{

   private int id;
   private String name;
   private Publisher publisher;

   public Book(){
   }

   @Id
   @GeneratedValue(strategy= GenerationType.IDENTITY)
   @Column(name="id")
   public int getId() {
      return id;
   }
```

```
   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }


   @Embedded
   @AttributeOverrides({
      @AttributeOverride(name = "name",
         column = @Column(name = "PUBLISHER")),
      @AttributeOverride(name = "address",
         column = @Column(name = "PUBLISHER_ADDRESS"))
   })
   public Publisher getPublisher() {
      return publisher;
   }

   public void setPublisher(Publisher publisher) {
      this.publisher = publisher;
   }
}
```

LibraryPersistentBeanRemote.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Remote;
```

```
@Remote
public interface LibraryPersistentBeanRemote {

   void addBook(Book bookName);

   List<Book> getBooks();

}
```

LibraryPersistentBean.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
public class LibraryPersistentBean implements LibraryPersistentBeanRemote {

   public LibraryPersistentBean(){
   }

   @PersistenceContext(unitName="EjbComponentPU")
   private EntityManager entityManager;

   public void addBook(Book book) {
      entityManager.persist(book);
   }

   public List<Book> getBooks() {
      return entityManager.createQuery("From Book").getResultList();
   }
```

```
}
```

- As soon as you deploy the EjbComponent project on JBOSS, notice the jboss log.
- JBoss has automatically created a JNDI entry for our session bean - **LibraryPersistentBean/remote**.
- We will be using this lookup string to get remote business object of type - **com.tutorialspoint.interceptor.LibraryPersistentBeanRemote**

JBoss Application Server Log Output

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:

   LibraryPersistentBean/remote - EJB3.x Default Remote Business Interface

   LibraryPersistentBean/remote-
com.tutorialspoint.interceptor.LibraryPersistentBeanRemote - EJB3.x Remote
Business Interface

16:30:02,723 INFO  [SessionSpecContainer] Starting
jboss.j2ee:jar=EjbComponent.jar,name=LibraryPersistentBean,service=EJB3

16:30:02,723 INFO  [EJBContainer] STARTED EJB:
com.tutorialspoint.interceptor.LibraryPersistentBeanRemote ejbName:
LibraryPersistentBean

16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:


   LibraryPersistentBean/remote - EJB3.x Default Remote Business Interface

   LibraryPersistentBean/remote-
com.tutorialspoint.interceptor.LibraryPersistentBeanRemote - EJB3.x Remote
Business Interface

...
```

EJBTester (EJB Client)

jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory

java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces

java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service.
- InitialContext object will be used to lookup stateless session bean.

EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateful.LibraryBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
```

```
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testEmbeddedObjects();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options \n1. Add Book\n2. Exit \nEnter Choice: ");
   }

   private void testEmbeddedObjects(){

      try {
         int choice = 1;

         LibraryPersistentBeanRemote libraryBean =
        (LibraryPersistentBeanRemote)
         ctx.lookup("LibraryPersistentBean/remote");

         while (choice != 2) {
            String bookName;
            String publisherName;
            String publisherAddress;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
```

```
            System.out.print("Enter publisher name: ");
            publisherName = brConsoleReader.readLine();
            System.out.print("Enter publisher address: ");
            publisherAddress = brConsoleReader.readLine();
            Book book = new Book();
            book.setName(bookName);
            book.setPublisher
           (new Publisher(publisherName,publisherAddress));

            libraryBean.addBook(book);
         } else if (choice == 2) {
            break;
         }
      }

      List<Book> booksList = libraryBean.getBooks();

      System.out.println("Book(s) entered so far: " + booksList.size());
      int i = 0;
      for (Book book:booksList) {
         System.out.println((i+1)+". " + book.getName());
         System.out.println("Publication: "+book.getPublisher());
         i++;
      }
   } catch (Exception e) {
      System.out.println(e.getMessage());
      e.printStackTrace();
   }finally {
      try {
         if(brConsoleReader !=null){
            brConsoleReader.close();
         }
      } catch (IOException ex) {
         System.out.println(ex.getMessage());
      }
   }
```

```
    }
}
```

EJBTester performs the following tasks:

- Load properties from jndi.properties and initialize the InitialContext object.
- In testInterceptedEjb() method, jndi lookup is done with the name - "LibraryPersistenceBean/remote" to obtain the remote business object (stateless EJB).
- Then the user is shown a library store User Interface and he/she is asked to enter a choice.
- If the user enters 1, the system asks for book name and saves the book using stateless session bean addBook() method. Session Bean is storing the book in database.
- If the user enters 2, the system retrieves books using stateless session bean getBooks() method and exits.

Run Client to Access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 1
Enter book name: learn html5
Enter publisher name: SAMS
Enter publisher address: DELHI
**********************
Welcome to Book Store
**********************
Options
```

```
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 1
1. learn html5
Publication: SAMS,DELHI
BUILD SUCCESSFUL (total time: 21 seconds)
```

14. EJB – BLOBS/CLOBS

EJB 3.0 provides support for Blob and Clob types using @Lob annotation. Following java types can be mapped using @Lob annotation.

- java.sql.Blob
- java.sql.Clob
- byte[]
- String
- Serializable Object

```
@Entity
@Table(name="books")
@EntityListeners(BookCallbackListener.class)
public class Book implements Serializable{
   ...
   private byte[] image;

   @Lob @Basic(fetch= FetchType.EAGER)
   public byte[] getImage() {
      return image;
   }
   ...
}
```

Example Application

Let us create a test EJB application to test blob/clob support in EJB 3.0.

Step	Description
1	Create a project with a name *EjbComponent* under a package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. Please use the project created in *EJB - Persistence* chapter as such for this chapter to understand clob/blob objects in ejb concepts.
2	Create *Book.java* under package *com.tutorialspoint.entity*. Use *EJB - Persistence* chapter as reference. Keep rest of the files unchanged.
3	Clean and Build the application to make sure business logic is working as per the requirements.

4	Finally, deploy the application in the form of a jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.
5	Now create the EJB client, a console based application in the same way as explained in the *EJB - Create Application* chapter under topic **Create Client to access EJB**.

Create/Alter Book Table

```
CREATE TABLE book (
   id     integer PRIMARY KEY,
   name   varchar(50)
);
Alter table book add image bytea;
Alter table book add xml text;
```

EJBComponent (EJB Module)

Book.java

```
package com.tutorialspoint.entity;

import com.tutorialspoint.callback.BookCallbackListener;
import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EntityListeners;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Lob;
import javax.persistence.Table;

@Entity
@Table(name="book")
public class Book implements Serializable{
```

```
private int id;
private String name;
private byte[] image;
private String xml;

public Book(){
}

@Id
@GeneratedValue(strategy= GenerationType.IDENTITY)
@Column(name="id")
public int getId() {
   return id;
}

public void setId(int id) {
   this.id = id;
}

public String getName() {
   return name;
}

public void setName(String name) {
   this.name = name;
}

@Lob @Basic(fetch= FetchType.EAGER)
public byte[] getImage() {
   return image;
}

public void setImage(byte[] image) {
   this.image = image;
}
```

```
   @Lob @Basic(fetch= FetchType.EAGER)
   public String getXml() {
      return xml;
   }

   public void setXml(String xml) {
      this.xml = xml;
   }
}
```

LibraryPersistentBeanRemote.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryPersistentBeanRemote {

   void addBook(Book bookName);

   List<Book> getBooks();

}
```

LibraryPersistentBean.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
```

```
@Stateless
public class LibraryPersistentBean implements LibraryPersistentBeanRemote {

   public LibraryPersistentBean(){
   }

   @PersistenceContext(unitName="EjbComponentPU")
   private EntityManager entityManager;

   public void addBook(Book book) {
      entityManager.persist(book);
   }

   public List<Book> getBooks() {
      return entityManager.createQuery("From Book").getResultList();
   }
}
```

- As soon as you deploy the EjbComponent project on JBOSS, notice the jboss log.
- JBoss has automatically created a JNDI entry for our session bean - **LibraryPersistentBean/remote**.
- We will using this lookup string to get remote business object of type - **com.tutorialspoint.interceptor.LibraryPersistentBeanRemote**

JBoss Application Server Log Output

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:
   LibraryPersistentBean/remote - EJB3.x Default Remote Business Interface
   LibraryPersistentBean/remote-
com.tutorialspoint.interceptor.LibraryPersistentBeanRemote - EJB3.x Remote
Business Interface
16:30:02,723 INFO  [SessionSpecContainer] Starting
jboss.j2ee:jar=EjbComponent.jar,name=LibraryPersistentBean,service=EJB3
16:30:02,723 INFO  [EJBContainer] STARTED EJB:
com.tutorialspoint.interceptor.LibraryPersistentBeanRemote ejbName:
LibraryPersistentBean
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:
```

```
   LibraryPersistentBean/remote - EJB3.x Default Remote Business Interface
   LibraryPersistentBean/remote-
com.tutorialspoint.interceptor.LibraryPersistentBeanRemote - EJB3.x Remote
Business Interface
...
```

EJBTester (EJB Client)

jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service.
- InitialContext object will be used to lookup stateless session bean.

EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateful.LibraryBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
```

```
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testBlobClob();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options \n1. Add Book\n2. Exit \nEnter Choice: ");
   }

   private void testBlobClob(){

      try {
         int choice = 1;

         LibraryPersistentBeanRemote libraryBean =
        (LibraryPersistentBeanRemote)
```

```
        ctx.lookup("LibraryPersistentBean/remote");

        while (choice != 2) {
           String bookName;
           String publisherName;
           String publisherAddress;
           showGUI();
           String strChoice = brConsoleReader.readLine();
           choice = Integer.parseInt(strChoice);
           if (choice == 1) {
              System.out.print("Enter book name: ");
              bookName = brConsoleReader.readLine();
              String xml = "<book><name>"+bookName+"</name></book>";
              Book book = new Book();
              book.setName(bookName);
              byte[] imageBytes = {0x32, 0x32,0x32, 0x32,0x32,
              0x32,0x32, 0x32,
              0x32, 0x32,0x32, 0x32,0x32, 0x32,0x32, 0x32,
              0x32, 0x32,0x32, 0x32,0x32, 0x32,0x32, 0x32
              };
              book.setImage(imageBytes);
              book.setXml(xml);

              libraryBean.addBook(book);
           } else if (choice == 2) {
              break;
           }
        }

        List<Book> booksList = libraryBean.getBooks();

        System.out.println("Book(s) entered so far: " + booksList.size());
        int i = 0;
        for (Book book:booksList) {
           System.out.println((i+1)+". " + book.getName());
           byte[] imageByts = book.getImage();
```

```
            if(imageByts != null){
               System.out.print("image bytes: [");
               for(int j = 0; j < imageByts.length ; j++){
                  System.out.print("0x"
                  + String.format("%x", imageByts[j]) +" ");
               }
               System.out.println("]");
            }
            System.out.println(book.getXml());
            i++;
         }
      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
   }
}
```

EJBTester performs the following tasks.

- Load properties from jndi.properties and initialize the InitialContext object.
- In testInterceptedEjb() method, jndi lookup is done with name - "LibraryPersistenceBean/remote" to obtain the remote business object (stateless EJB).
- Then the user is shown a library store User Interface and he/she is asked to enter a choice.
- If the user enters 1, the system asks for book name and saves the book using stateless session bean addBook() method. Session Bean is storing the book in database.
- If the user enters 2, the system retrieves books using stateless session bean getBooks() method and exits.

Run Client to Access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 1
Enter book name: learn testing
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 1
1. learn testing
image bytes: [
0x32 0x32 0x32 0x32 0x32 0x32 0x32 0x32 0x32 0x32 0x32 0x32 0x32 0x32 0x32 0x32
0x32 0x32 0x32 0x32 0x32 0x32 0x32 0x32 ]
<book><name>learn testing</name></book>
BUILD SUCCESSFUL (total time: 20 seconds)
```

15. EJB – TRANSACTIONS

A transaction is a single unit of work items, which follows the ACID properties. ACID stands for Atomic, Consistent, Isolated, and Durable.

- **Atomic** - If any of the work item fails, the whole unit will be considered failed. Success meant, all items execute successfully.
- **Consistent** - A transaction must keep the system in consistent state.
- **Isolated** - Each transaction executes independent of any other transaction.
- **Durable** - Transaction should survive system failure if it has been executed or committed.

EJB Container/Servers are transaction servers and handles transactions context propagation and distributed transactions. Transactions can be managed by the container or by custom code handling in bean's code.

- **Container Managed Transactions** - In this type, the container manages the transaction states.
- **Bean Managed Transactions** - In this type, the developer manages the life cycle of transaction states.

Container Managed Transactions

EJB 3.0 has specified following attributes of transactions, which EJB containers implement:

- **REQUIRED** - Indicates that business method has to be executed within transaction, otherwise a new transaction will be started for that method.
- **REQUIRES_NEW** - Indicates that a new transaction is to be started for the business method.
- **SUPPORTS** - Indicates that business method will execute as part of transaction.
- **NOT_SUPPORTED** - Indicates that business method should not be executed as part of transaction.
- **MANDATORY** - Indicates that business method will execute as part of transaction, otherwise exception will be thrown.
- **NEVER** - Indicates if business method executes as part of transaction, then an exception will be thrown.

Example

```
package com.tutorialspoint.txn.required;

import javax.ejb.*

@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
public class UserDetailBean implements UserDetailRemote {

   private UserDetail;

   @TransactionAttribute(TransactionAttributeType.REQUIRED)
   public void createUserDetail() {
      //create user details object
   }
}
```

createUserDetail() business method is made Required using Required annotation.

```
package com.tutorialspoint.txn.required;

import javax.ejb.*

@Stateless
public class UserSessionBean implements UserRemote {

   private User;

   @EJB
   private UserDetailRemote userDetail;

   public void createUser() {
      //create user
      //...
      //create user details
      userDetail.createUserDetail();
   }
```

```
}
```

createUser() business method is using createUserDetail(). If exception occurred during createUser() call and User object is not created then UserDetail object will also not be created.

Bean Managed Transactions

In Bean Managed Transactions, Transactions can be managed by handling exceptions at application level.

Following are the key points to be considered:

- **Start** - When to start a transaction in a business method.
- **Success** - Identify success scenario when a transaction is to be committed.
- **Failed** - Identify failure scenario when a transaction is to be rollback.

Example

```
package com.tutorialspoint.txn.bmt;

import javax.annotation.Resource;
import javax.ejb.Stateless;
import javax.ejb.TransactionManagement;
import javax.ejb.TransactionManagementType;
import javax.transaction.UserTransaction;

@Stateless
@TransactionManagement(value=TransactionManagementType.BEAN)
public class AccountBean implements AccountBeanLocal {

   @Resource
   private UserTransaction userTransaction;

   public void transferFund(Account fromAccount, double fund ,
      Account toAccount) throws Exception{

      try{
         userTransaction.begin();

         confirmAccountDetail(fromAccount);
```

```
            withdrawAmount(fromAccount,fund);

            confirmAccountDetail(toAccount);
            depositAmount(toAccount,fund);

            userTransaction.commit();
         }catch (InvalidAccountException exception){
            userTransaction.rollback();
         }catch (InsufficientFundException exception){
            userTransaction.rollback();
         }catch (PaymentException exception){
            userTransaction.rollback();
         }
      }

      private void confirmAccountDetail(Account account)
         throws InvalidAccountException {
      }

      private void withdrawAmount() throws InsufficientFundException {
      }

      private void depositAmount() throws PaymentException{
      }
   }
```

In this example, we made use of **UserTransaction** interface to mark the beginning of transaction using **userTransaction.begin()** method call. We mark the completion of transaction by using **userTransaction.commit()** method and if any exception occurred during transaction, then we rollback the complete transaction using **userTransaction.rollback()** method call.

16. EJB – SECURITY

Security is a major concern of any enterprise level application. It includes identification of user(s) or system accessing the application. Based on identification, it allows or denies the access to resources within the application. An EJB container manages standard security concerns or it can be customized to handle any specific security concerns.

Important Terms of Security

- **Authentication** - This is the process ensuring that user accessing the system or application is verified to be authentic.
- **Authorization** - This is the process ensuring that authentic user has right level of authority to access system resources.
- **User** - User represents the client or system, which accesses the application.
- **User Groups** - Users may be part of the group having certain authorities. For example, administrator's group.
- **User Roles** - Roles define the level of authority, a user have or permissions to access a system resource.

Container Managed Security

EJB 3.0 has specified following attributes/annotations of security, which EJB containers implement.

- **DeclareRoles** - Indicates that class will accept the declared roles. Annotations are applied at class level.
- **RolesAllowed** - Indicates that a method can be accessed by the user of specified role. It can be applied at class level where all methods of a class can be accessed by the user of specified role.
- **PermitAll** - Indicates that a business method is accessible to all. It can be applied at class as well as at method level.
- **DenyAll** - Indicates that a business method is not accessible to any of the user specified at class or at method level.

Example

```
package com.tutorialspoint.security.required;


import javax.ejb.*


@Stateless
```

```
@DeclareRoles({"student" "librarian"})
public class LibraryBean implements LibraryRemote {

   @RolesAllowed({"librarian"})
   public void delete(Book book){
       //delete book
   }

   @PermitAll
   public void viewBook(Book book){
      //view book
   }

   @DenyAll
   public void deleteAll(){
      //delete all books
   }
}
```

Security Configuration

Map roles and user grouped in configuration file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Application Server
9.0 EJB 3.0//EN" "http://www.sun.com/software/appserver/dtds/sun-ejb-jar_3_0-
0.dtd">
<ejb-jar>
   <security-role-mapping>
      <role-name>student</role-name>
      <group-name>student-group</group-name>
   </security-role-mapping>
   <security-role-mapping>
      <role-name>librarian</role-name>
      <group-name>librarian-group</group-name>
   </security-role-mapping>
   <enterprise-beans/>
```

```
</ejb-jar>
```

17. EJB – JNDI BINDINGS

JNDI stands for Java Naming and Directory Interface. It is a set of API and service interfaces. Java based applications use JNDI for naming and directory services. In context of EJB, there are two terms.

- **Binding** - This refers to assigning a name to an EJB object, which can be used later.
- **Lookup** - This refers to looking up and getting an object of EJB.

In Jboss, session beans are bound in JNDI in the following format by default.

- **local** - EJB-name/local
- **remote** - EJB-name/remote

In case, EJB are bundled with <application-name>.ear file, then default format is as following:

- **local** - application-name/ejb-name/local
- **remote** - application-name/ejb-name/remote

Example of Default Binding

Refer to *EJB - Create Application* chapter's JBoss console output.

JBoss Application Server Log Output

```
...
16:30:02,723 INFO  [SessionSpecContainer] Starting
jboss.j2ee:jar=EjbComponent.jar,name=LibrarySessionBean,service=EJB3
16:30:02,723 INFO  [EJBContainer] STARTED EJB:
com.tutorialspoint.stateless.LibrarySessionBean ejbName: LibrarySessionBean
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:


   LibrarySessionBean/remote - EJB3.x Default Remote Business Interface
   LibrarySessionBean/remote-
com.tutorialspoint.stateless.LibrarySessionBeanRemote - EJB3.x Remote Business
Interface
...
```

Customized Binding

Following annotations can be used to customize the default JNDI bindings:

- **local** - org.jboss.ejb3.LocalBinding
- **remote** - org.jboss.ejb3.RemoteBindings

Update LibrarySessionBean.java. Refer to *EJB - Create Application* chapter.

LibrarySessionBean

```
package com.tutorialspoint.stateless;
import java.util.ArrayList;
import java.util.List;
import javax.ejb.Stateless;

@Stateless
@LocalBinding(jndiBinding="tutorialsPoint/librarySession")
public class LibrarySessionBean implements LibrarySessionBeanLocal {

    List<String> bookShelf;
    public LibrarySessionBean(){
       bookShelf = new ArrayList<String>();
    }
    public void addBook(String bookName) {
       bookShelf.add(bookName);
    }

    public List<String> getBooks() {
        return bookShelf;
    }
}
```

LibrarySessionBeanLocal

```
package com.tutorialspoint.stateless;


import java.util.List;

import javax.ejb.Local;

```

```
@Local
public interface LibrarySessionBeanLocal {

    void addBook(String bookName);

    List getBooks();

}
```

Build the project, deploy the application on Jboss, and verify the following output in Jboss console:

```
...
16:30:02,723 INFO  [SessionSpecContainer] Starting
jboss.j2ee:jar=EjbComponent.jar,name=LibrarySessionBean,service=EJB3
16:30:02,723 INFO  [EJBContainer] STARTED EJB:
com.tutorialspoint.stateless.LibrarySessionBean ejbName: LibrarySessionBean
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:

   tutorialsPoint/librarySession - EJB3.x Default Local Business Interface
   tutorialsPoint/librarySession-
com.tutorialspoint.stateless.LibrarySessionBeanLocal - EJB3.x Local Business
Interface
...
```

18. EJB – ENTITY RELATIONSHIPS

EJB 3.0 provides option to define database entity relationships/mappings like one-to-one, one-to-many, many-to-one, and many-to-many relationships.

Following are the relevant annotations:

- **One-to-One** - Objects have one-to-one relationship. For example, a passenger can travel using a single ticket at a time.
- **One-to-Many** - Objects have one-to-many relationship. For example, a father can have multiple kids.
- **Many-to-One** - Objects have many-to-one relationship. For example, multiple kids having a single mother.
- **Many-to-Many** - Objects have many-to-many relationship. For example, a book can have multiple authors and an author can write multiple books.

We will demonstrate use of ManyToMany mapping here. To represent ManyToMany relationship, three following tables are required:

- **Book** - Book table, having records of books.
- **Author** - Author table, having records of author.
- **Book_Author** – Book_Author table, having linkage of above mentioned Book and Author table.

Create Tables

Create a table **book author**, **book_author** in default database **postgres**.

```
CREATE TABLE book (
   book_id     integer,
   name    varchar(50)
);
CREATE TABLE author (
   author_id   integer,
   name    varchar(50)
);
CREATE TABLE book_author (
   book_id     integer,
   author_id   integer
);
```

Create Entity Classes

```
@Entity
@Table(name="author")
public class Author implements Serializable{
   private int id;
   private String name;
   ...
}
@Entity
@Table(name="book")
public class Book implements Serializable{
   private int id;
   private String title;
   private Set<Author> authors;
   ...
}
```

Use ManyToMany annotation in Book Entity.

```
@Entity
public class Book implements Serializable{
   ...
   @ManyToMany(cascade = {CascadeType.PERSIST, CascadeType.MERGE}
      , fetch = FetchType.EAGER)
   @JoinTable(table = @Table(name = "book_author"),
      joinColumns = {@JoinColumn(name = "book_id")},
      inverseJoinColumns = {@JoinColumn(name = "author_id")})
   public Set<Author> getAuthors()
   {
      return authors;
   }
   ...
}
```

Example Application

Let us create a test EJB application to test entity relationships objects in EJB 3.0.

Step	Description
1	Create a project with a name *EjbComponent* under a package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. Please use the project created in *EJB - Persistence* chapter as such for this chapter to understand embedded objects in EJB concepts.
2	Create *Author.java* under package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. Keep rest of the files unchanged.
3	Create *Book.java* under package *com.tutorialspoint.entity*. Use *EJB - Persistence* chapter as reference. Keep rest of the files unchanged.
4	Clean and Build the application to make sure business logic is working as per the requirements.
5	Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.
6	Now create the EJB client, a console based application in the same way as explained in the *EJB - Create Application* chapter under topic **Create Client to access EJB**.

EJBComponent (EJB Module)

Author.java

```
package com.tutorialspoint.entity;


import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;


@Entity
@Table(name="author")
public class Author implements Serializable{


   private int id;
   private String name;
```

```
    public Author(){}

    public Author(int id, String name){
       this.id = id;
       this.name = name;
    }

    @Id
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    @Column(name="author_id")
    public int getId() {
       return id;
    }

    public void setId(int id) {
       this.id = id;
    }

    public String getName() {
       return name;
    }

    public void setName(String name) {
       this.name = name;
    }

    public String toString(){
       return id + "," + name;
    }
}
```

Book.java

```
package com.tutorialspoint.entity;


import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Table;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;

@Entity
@Table(name="book")
public class Book implements Serializable{

   private int id;
   private String name;
   private Set<Author> authors;

   public Book(){
   }

   @Id
   @GeneratedValue(strategy= GenerationType.IDENTITY)
   @Column(name="book_id")
   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
```

```
   }

   public void setName(String name) {
      this.name = name;
   }

   public void setAuthors(Set<Author> authors) {
      this.authors = authors;
   }

   @ManyToMany(cascade = {CascadeType.PERSIST, CascadeType.MERGE}
      , fetch = FetchType.EAGER)
   @JoinTable(table = @Table(name = "book_author"),
      joinColumns = {@JoinColumn(name = "book_id")},
      inverseJoinColumns = {@JoinColumn(name = "author_id")})
   public Set<Author> getAuthors()
   {
      return authors;
   }
}
```

LibraryPersistentBeanRemote.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryPersistentBeanRemote {

   void addBook(Book bookName);

   List<Book> getBooks();
```

```
}
```

LibraryPersistentBean.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
public class LibraryPersistentBean implements LibraryPersistentBeanRemote {

   public LibraryPersistentBean(){
   }

   @PersistenceContext(unitName="EjbComponentPU")
   private EntityManager entityManager;

   public void addBook(Book book) {
      entityManager.persist(book);
   }

   public List<Book> getBooks() {
      return entityManager.createQuery("From Book").getResultList();
   }
}
```

- As soon as you deploy the EjbComponent project on JBOSS, notice the jboss log.
- JBoss has automatically created a JNDI entry for our session bean - **LibraryPersistentBean/remote**.
- We will using this lookup string to get remote business object of type - **com.tutorialspoint.interceptor.LibraryPersistentBeanRemote**

JBoss Application Server Log Output

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:

   LibraryPersistentBean/remote - EJB3.x Default Remote Business Interface

   LibraryPersistentBean/remote-
com.tutorialspoint.interceptor.LibraryPersistentBeanRemote - EJB3.x Remote
Business Interface

16:30:02,723 INFO  [SessionSpecContainer] Starting
jboss.j2ee:jar=EjbComponent.jar,name=LibraryPersistentBean,service=EJB3

16:30:02,723 INFO  [EJBContainer] STARTED EJB:
com.tutorialspoint.interceptor.LibraryPersistentBeanRemote ejbName:
LibraryPersistentBean

16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:


   LibraryPersistentBean/remote - EJB3.x Default Remote Business Interface

   LibraryPersistentBean/remote-
com.tutorialspoint.interceptor.LibraryPersistentBeanRemote - EJB3.x Remote
Business Interface

...
```

EJBTester (EJB Client)

jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory

java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces

java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service.
- InitialContext object will be used to lookup stateless session bean.

EJBTester.java

```
package com.tutorialspoint.test;


import com.tutorialspoint.stateful.LibraryBeanRemote;

import java.io.BufferedReader;

import java.io.FileInputStream;
```

```
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.*;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testEmbeddedObjects();
   }

   private void showGUI(){
      System.out.println("**********************");
```

```
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options \n1. Add Book\n2. Exit \nEnter Choice: ");
   }

   private void testEmbeddedObjects(){

      try {
         int choice = 1;

         LibraryPersistentBeanRemote libraryBean =
         (LibraryPersistentBeanRemote)
         ctx.lookup("LibraryPersistentBean/remote");

         while (choice != 2) {
            String bookName;
            String authorName;

            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               System.out.print("Enter author name: ");
               authorName = brConsoleReader.readLine();
               Book book = new Book();
               book.setName(bookName);
                     Author author = new Author();
                     author.setName(authorName);
                     Set<Author> authors = new HashSet<Author>();
                     authors.add(author);
               book.setAuthors(authors);

               libraryBean.addBook(book);
            } else if (choice == 2) {
```

```
                break;
            }
         }

         List<Book> booksList = libraryBean.getBooks();

         System.out.println("Book(s) entered so far: " + booksList.size());
         int i = 0;
         for (Book book:booksList) {
            System.out.println((i+1)+". " + book.getName());
            System.out.print("Author: ");
            Author[] authors = (Author[])books.getAuthors().toArray();
            for(int j=0;j<authors.length;j++){
               System.out.println(authors[j]);
            }
            i++;
         }
      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
   }
}
```

EJBTester performs the following tasks:

- Load properties from jndi.properties and initialize the InitialContext object.
- In testInterceptedEjb() method, jndi lookup is done with name - "LibraryPersistenceBean/remote" to obtain the remote business object (stateless EJB).
- Then the user is shown a library store User Interface and he/she is asked to enter a choice.
- If the user enters 1, the system asks for book name and saves the book using stateless session bean addBook() method. Session Bean is storing the book in database.
- If the user enters 2, the system retrieves books using stateless session bean getBooks() method and exits.

Run Client to Access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 1
Enter book name: learn html5
Enter Author name: Robert
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 1
1. learn html5
Author: Robert
```

```
BUILD SUCCESSFUL (total time: 21 seconds)
```

19. EJB – ACCESS DATABASE

In EJB 3.0, persistence mechanism is used to access the database in which the container manages the database related operations. Developers can access database using JDBC API call directly in EJB business methods.

To demonstrate database access in EJB, we need to perform the following tasks:

- Step 1. Create a table in the database.
- Step 2. Create a stateless EJB having business me.
- Step 3. Update stateless EJB. Add methods to add records and get records from database via entity manager.
- Step 4. A console based application client will access the stateless EJB to persist data in database.

Create Table

Create a table **books** in default database **postgres**.

```
CREATE TABLE books (
   id     integer PRIMARY KEY,
   name   varchar(50)
);
```

Create a Model Class

```
public class Book implements Serializable{

   private int id;
   private String name;

   public Book(){
   }

   public int getId() {
      return id;
   }
   ...
```

```
}
```

Create Stateless EJB

```
@Stateless
public class LibraryPersistentBean implements LibraryPersistentBeanRemote {

   public void addBook(Book book) {
     //persist book using jdbc calls
   }

   public List<Book> getBooks() {
     //get books using jdbc calls
   }
   ...
}
```

After building the EJB module, we need a client to access the stateless bean, which we will be going to create in the next section.

Example Application

Let us create a test EJB application to test EJB database access mechanism.

Step	Description
1	Create a project with a name *EjbComponent* under a package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. You can also use the project created in *EJB - Create Application* chapter as such for this chapter to understand EJB data access concepts.
2	Create *Book.java* under package *com.tutorialspoint.entity* and modify it as shown below.
3	Create *LibraryPersistentBean.java* and *LibraryPersistentBeanRemote* as explained in the *EJB - Create Application* chapter and modify them as shown below.
4	Clean and Build the application to make sure business logic is working as per the requirements.
5	Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.
6	Now create the EJB client, a console based application in the same way as explained in the *EJB - Create Application* chapter under topic **Create Client to access EJB**. Modify it as shown below.

EJBComponent (EJB Module)

Book.java

```
package com.tutorialspoint.entity;

import java.io.Serializable;

public class Book implements Serializable{

   private int id;
   private String name;

   public Book(){
   }

   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
}
```

LibraryPersistentBeanRemote.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryPersistentBeanRemote {

   void addBook(Book bookName);

   List<Book> getBooks();

}
```

LibraryPersistentBean.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import javax.ejb.Stateless;

@Stateless
public class LibraryPersistentBean implements LibraryPersistentBeanRemote {

   public LibraryPersistentBean(){
   }
```

```
public void addBook(Book book) {
   Connection con = null;
   String url = "jdbc:postgresql://localhost:5432/postgres";
   String driver = "org.postgresql.driver";

   String userName = "sa";
   String password = "sa";
   List<Book> books = new ArrayList<Book>();
   try {

      Class.forName(driver).newInstance();
      con = DriverManager.getConnection(url , userName, password);

      PreparedStatement st =
      con.prepareStatement("insert into book(name) values(?)");
      st.setString(1,book.getName());

      int result = st.executeUpdate();

   } catch (SQLException ex) {
      ex.printStackTrace();
   } catch (InstantiationException ex) {
      ex.printStackTrace();
   } catch (IllegalAccessException ex) {
      ex.printStackTrace();
   } catch (ClassNotFoundException ex) {
      ex.printStackTrace();
   }
}

public List<Book> getBooks() {
   Connection con = null;
   String url = "jdbc:postgresql://localhost:5432/postgres";
   String driver = "org.postgresql.driver";
```

```
      String userName = "sa";
      String password = "sa";
      List<Book> books = new ArrayList<Book>();
      try {

         Class.forName(driver).newInstance();
         con = DriverManager.getConnection(url , userName, password);

         Statement st = con.createStatement();
         ResultSet rs = st.executeQuery("select * from book");

         Book book;
         while (rs.next()) {
            book = new Book();
            book.setId(rs.getInt(1));
            book.setName(rs.getString(2));
            books.add(book);
         }
      } catch (SQLException ex) {
         ex.printStackTrace();
      } catch (InstantiationException ex) {
         ex.printStackTrace();
      } catch (IllegalAccessException ex) {
         ex.printStackTrace();
      } catch (ClassNotFoundException ex) {
         ex.printStackTrace();
      }
      return books;
   }
}
```

- As soon as you deploy the EjbComponent project on JBOSS, notice the jboss log.
- JBoss has automatically created a JNDI entry for our session bean - **LibraryPersistentBean/remote**.
- We will be using this lookup string to get remote business object of type - **com.tutorialspoint.stateless.LibraryPersistentBeanRemote**

JBoss Application Server Log Output

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:

   LibraryPersistentBean/remote - EJB3.x Default Remote Business Interface

   LibraryPersistentBean/remote-
com.tutorialspoint.stateless.LibraryPersistentBeanRemote - EJB3.x Remote
Business Interface

16:30:02,723 INFO  [SessionSpecContainer] Starting
jboss.j2ee:jar=EjbComponent.jar,name=LibraryPersistentBeanRemote,service=EJB3

16:30:02,723 INFO  [EJBContainer] STARTED EJB:
com.tutorialspoint.stateless.LibraryPersistentBeanRemote ejbName:
LibraryPersistentBean

16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:


   LibraryPersistentBean/remote - EJB3.x Default Remote Business Interface

   LibraryPersistentBean/remote-
com.tutorialspoint.stateless.LibraryPersistentBeanRemote - EJB3.x Remote
Business Interface

...
```

EJBTester (EJB Client)

jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory

java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces

java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service.
- InitialContext object will be used to lookup stateless session bean.

EJBTester.java

```
package com.tutorialspoint.test;


import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;

import java.io.BufferedReader;

import java.io.FileInputStream;
```

```
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testEntityEjb();
   }

   private void showGUI(){
```

```
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options \n1. Add Book\n2. Exit \nEnter Choice: ");
   }

   private void testEntityEjb(){

      try {
         int choice = 1;

         LibraryPersistentBeanRemote libraryBean =
         LibraryPersistentBeanRemote)
         ctx.lookup("LibraryPersistentBean/remote");

         while (choice != 2) {
            String bookName;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               Book book = new Book();
               book.setName(bookName);
               libraryBean.addBook(book);
            } else if (choice == 2) {
               break;
            }
         }

         List<Book> booksList = libraryBean.getBooks();

         System.out.println("Book(s) entered so far: " + booksList.size());
         int i = 0;
         for (Book book:booksList) {
```

```
            System.out.println((i+1)+". " + book.getName());
            i++;
         }
      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
   }
}
```

EJBTester performs the following tasks:

- Load properties from jndi.properties and initialize the InitialContext object.
- In testStatefulEjb() method, jndi lookup is done with the name - "LibraryStatelessSessionBean/remote" to obtain the remote business object (stateful EJB).
- Then user is shown a library store User Interface and he/she is asked to enter a choice.
- If the user enters 1, the system asks for book name and saves the book using stateless session bean addBook() method. Session Bean is persisting the book in database via EntityManager call.
- If the user enters 2, the system retrieves books using stateless session bean getBooks() method and exits.
- Then another jndi lookup is done with the name - "LibraryStatelessSessionBean/remote" to obtain the remote business object (stateful EJB) again and listing of books is done.

Run Client to Access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 1
Enter book name: Learn Java
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 1
1. learn java
BUILD SUCCESSFUL (total time: 15 seconds)
```

20. EJB – QUERY LANGUAGE

EJB Query Language is quite handy to write custom queries without worrying about underlying database details. It is quite similar to HQL, hibernate query language and is often referred by the name EJBQL.

To demonstrate EJBQL in EJB, we are going to do the following tasks:

- Step 1. Create table in database.
- Step 2. Create a stateless EJB having business name.
- Step 3. Update stateless EJB. Add methods to add records and get records from database via entity manager.
- Step 4. A console based application client will access the stateless EJB to persist data in database.

Create Table

Create a table **books** in default database **postgres**.

```
CREATE TABLE books (
   id     integer PRIMARY KEY,
   name   varchar(50)
);
```

Create a Model Class

```
public class Book implements Serializable{

   private int id;
   private String name;

   public Book(){
   }

   public int getId() {
      return id;
   }
   ...
```

```
}
```

Create Stateless EJB

```
@Stateless
public class LibraryPersistentBean implements LibraryPersistentBeanRemote {

   public void addBook(Book book) {
     //persist book using entity manager
   }

   public List<Book> getBooks() {
     //get books using entity manager
   }
   ...
}
```

After building the EJB module, we need a client to access the stateless bean, which we will be going to create in the next section.

Example Application

Let us create a test EJB application to test EJB database access mechanism.

Step	Description
1	Create a project with a name *EjbComponent* under a package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. You can also use the project created in *EJB - Create Application* chapter as such for this chapter to understand EJB data access concepts.
2	Create *Book.java* under package *com.tutorialspoint.entity* and modify it as shown below.
3	Create *LibraryPersistentBean.java* and *LibraryPersistentBeanRemote* as explained in the *EJB - Create Application* chapter and modify them as shown below.
4	Clean and Build the application to make sure business logic is working as per the requirements.
5	Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.
6	Now create the EJB client, a console based application in the same way as explained in the *EJB - Create Application* chapter under topic **Create Client to access EJB**. Modify it as shown below.

EJBComponent (EJB Module)

Book.java

```
package com.tutorialspoint.entity;

import java.io.Serializable;

public class Book implements Serializable{

   private int id;
   private String name;

   public Book(){
   }

   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
}
```

LibraryPersistentBeanRemote.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryPersistentBeanRemote {

   void addBook(Book bookName);

   List<Book> getBooks();

}
```

LibraryPersistentBean.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;

@Stateless
public class LibraryPersistentBean implements LibraryPersistentBeanRemote {

   public LibraryPersistentBean(){
   }

   @PersistenceContext(unitName="EntityEjbPU")
   private EntityManager entityManager;
```

```
   public void addBook(Book book) {
      entityManager.persist(book);
   }

   public List<Book> getBooks() {
      //create an ejbql expression
      String ejbQL = "From Book b where b.name like ?1";
      //create query
      Query query = entityManager.createQuery(ejbQL);
      //substitute parameter.
      query.setParameter(1, "%test%");
      //execute the query
      return query.getResultList();
   }
}
```

- As soon as you deploy the EjbComponent project on JBOSS, notice the jboss log.
- JBoss has automatically created a JNDI entry for our session bean - **LibraryPersistentBean/remote**.
- We will be using this lookup string to get remote business object of type - **com.tutorialspoint.stateless.LibraryPersistentBeanRemote**

JBoss Application Server Log Output

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:
   LibraryPersistentBean/remote - EJB3.x Default Remote Business Interface
   LibraryPersistentBean/remote-
com.tutorialspoint.stateless.LibraryPersistentBeanRemote - EJB3.x Remote
Business Interface
16:30:02,723 INFO  [SessionSpecContainer] Starting
jboss.j2ee:jar=EjbComponent.jar,name=LibraryPersistentBeanRemote,service=EJB3
16:30:02,723 INFO  [EJBContainer] STARTED EJB:
com.tutorialspoint.stateless.LibraryPersistentBeanRemote ejbName:
LibraryPersistentBean
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:

   LibraryPersistentBean/remote - EJB3.x Default Remote Business Interface
```

```
   LibraryPersistentBean/remote-
com.tutorialspoint.stateless.LibraryPersistentBeanRemote - EJB3.x Remote
Business Interface

...
```

EJBTester (EJB Client)

jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory

java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces

java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service.
- InitialContext object will be used to lookup stateless session bean.

EJBTester.java

```
package com.tutorialspoint.test;


import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;

import java.io.BufferedReader;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.List;

import java.util.Properties;

import javax.naming.InitialContext;

import javax.naming.NamingException;


public class EJBTester {


   BufferedReader brConsoleReader = null;

   Properties props;

   InitialContext ctx;

   {

      props = new Properties();

      try {
```

```
            props.load(new FileInputStream("jndi.properties"));
         } catch (IOException ex) {
            ex.printStackTrace();
         }
         try {
            ctx = new InitialContext(props);
         } catch (NamingException ex) {
            ex.printStackTrace();
         }
         brConsoleReader =
         new BufferedReader(new InputStreamReader(System.in));
      }

      public static void main(String[] args) {

         EJBTester ejbTester = new EJBTester();

         ejbTester.testEntityEjb();
      }

      private void showGUI(){
         System.out.println("**********************");
         System.out.println("Welcome to Book Store");
         System.out.println("**********************");
         System.out.print("Options \n1. Add Book\n2. Exit \nEnter Choice: ");
      }

      private void testEntityEjb(){

         try {
            int choice = 1;

            LibraryPersistentBeanRemote libraryBean =
            LibraryPersistentBeanRemote)
            ctx.lookup("LibraryPersistentBean/remote");
```

```
         while (choice != 2) {
            String bookName;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               Book book = new Book();
               book.setName(bookName);
               libraryBean.addBook(book);
            } else if (choice == 2) {
               break;
            }
         }

         List<Book> booksList = libraryBean.getBooks();

         System.out.println("Book(s) entered so far: " + booksList.size());
         int i = 0;
         for (Book book:booksList) {
            System.out.println((i+1)+". " + book.getName());
            i++;
         }
      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
```

```
    }
}
```

EJBTester performs the following tasks:

- Load properties from jndi.properties and initialize the InitialContext object.
- In testStatefulEjb() method, jndi lookup is done with the name - "LibraryStatelessSessionBean/remote" to obtain the remote business object (stateful ejb).
- Then user is shown a library store User Interface and he/she is asked to enter a choice.
- If user enters 1, system asks for book name and saves the book using stateless session bean addBook() method. Session Bean is persisting the book in database via EntityManager call.
- If user enters 2, system retrieves books using stateless session bean getBooks() method and exits.
- Then another jndi lookup is done with name - "LibraryStatelessSessionBean/remote" to obtain the remote business object (stateful EJB) again and listing of books is done.

Run Client to Access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 1
Enter book name: Learn Testing
**********************
Welcome to Book Store
**********************
Options
```

```
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 1
1. learn Testing
BUILD SUCCESSFUL (total time: 15 seconds)
```

21. EJB – EXCEPTION HANDLING

EJBs are a part of enterprise applications which are normally based on distributed environments. So, apart from the normal exceptions that can occur, there can be exceptions like communication failure, security permissions, server down, etc.

EJB container considers exceptions in two ways:

- **Application Exception** - If business rule is violated or exception occurs while executing the business logic.
- **System Exception**- Any exception, which is not caused by business logic or business code. For example, error during EJB lookup. RuntimeException, RemoteException are SystemException.

How Does EJB Container Handle Exceptions?

When **Application Exception** occurs, EJB container intercepts the exception, but returns the same to the client as it is. It does not roll back the transaction unless it is specified in the code by EJBContext.setRollBackOnly() method. EJB Container does not wrap the exception in case of Application Exception.

When **System Exception** occurs, EJB container intercepts the exception, rollbacks the transaction and start the cleanup tasks. It wraps the exception into RemoteException and throws it to the client.

Handling Application Exception

Application exceptions are generally thrown in Session EJB methods as these are the methods responsible to execute business logic. Application exception should be declared in throws clause of business method and should be thrown in case business logic fails.

```
@Stateless
public class LibraryPersistentBean implements LibraryPersistentBeanRemote {

   ...

   public List<Book> getBooks() throws NoBookAvailableException {
      List<Book> books =
         entityManager.createQuery("From Books").getResultList();
       if(books.size == 0)
           throw NoBookAvailableException
          ("No Book available in library.");
```

```
        return books;
    }
    ...
}
```

Handling System Exception

System exception can occur at any time like naming lookup fails, sql error occurs while fetching data. In such a case, such exception should be wrapped under EJBException and thrown back to the client.

```
@Stateless
public class LibraryPersistentBean implements LibraryPersistentBeanRemote {


   ...
   public List<Book> getBooks() {
      try {
         List<Book> books =
            entityManager.createQuery("From Books").getResultList();
      } catch (CreateException ce){
         throw (EJBException) new EJBException(ce).initCause(ce);
       } catch (SqlException se){
         throw (EJBException) new EJBException(se).initCause(se);
      }
       return books;
   }
   ...
}
```

At client side, handle the EJBException.

```
public class EJBTester {
   private void testEntityEjb(){
   ...
   try{
      LibraryPersistentBeanRemote libraryBean =
      LibraryPersistentBeanRemote)ctx.lookup("LibraryPersistentBean/remote");
```

```
      List<Book> booksList = libraryBean.getBooks();
   } catch(EJBException e) {
      Exception ne = (Exception) e.getCause();
      if(ne.getClass().getName().equals("SqlException")){
          System.out.println("Database error: "+ e.getMessage());
      }
   }
   ...
   }
```

22. EJB – WEB SERVICES

EJB 3.0 provides an option to expose session EJB as a webservice. @WebService annotation is used to mark a class as a web service end point and @WebMethod is used to expose a method as web method to client.

```
@Stateless
@WebService(serviceName="LibraryService")
public class LibraryPersistentBean implements LibraryPersistentBeanRemote {


   ...
   @WebMethod(operationName="getBooks")
   public List getBooks()  {
      return entityManager.createQuery("From Books").getResultList();
   }
   ...
}
```

Example Application

Let us create a test EJB application to test blob/clob support in EJB 3.0.

Step	Description
1	Create a project with a name EjbComponent under a package com.tutorialspoint.entity as explained in the EJB - Create Application chapter. Please use the project created in EJB - Persistence chapter as such for this chapter to understand clob/blob objects in EJB concepts.
2	Create LibraryPersistentBean.java under package com.tutorialspoint.stateless. Use EJB - Persistence chapter as reference. Keep rest of the files unchanged.
3	Clean and Build the application to make sure business logic is working as per the requirements.
4	Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.

LibraryPersistentBean.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Stateless;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
@WebService(serviceName="LibraryService")
public class LibraryPersistentBean implements LibraryPersistentBeanRemote {

   public LibraryPersistentBean(){
   }

   @PersistenceContext(unitName="EjbComponentPU")
   private EntityManager entityManager;

   public void addBook(Book book) {
      entityManager.persist(book);
   }

   @WebMethod(operationName="getBooks")
   public List <Book> getBooks() {
      return entityManager.createQuery("From Book").getResultList();
   }
}
```

JBoss Application Server Log Output

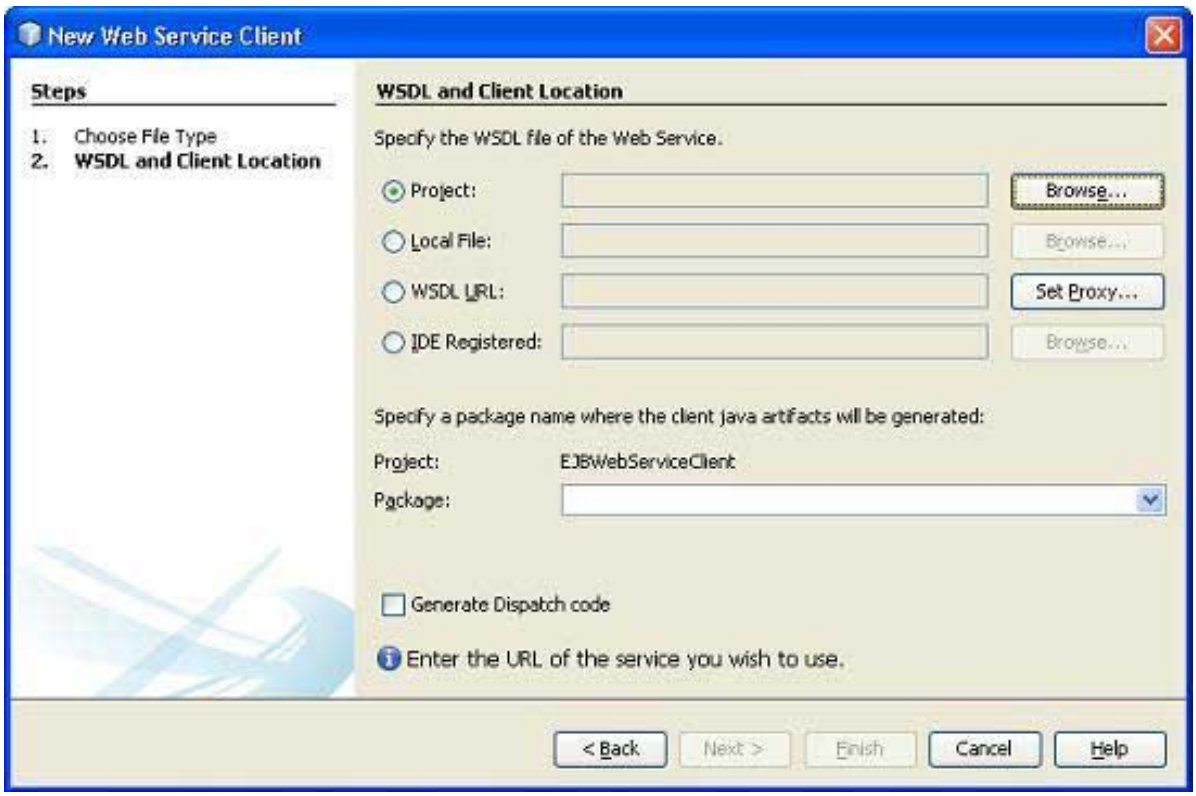
```
10:51:37,271 INFO  [EJBContainer] STARTED EJB:
com.tutorialspoint.stateless.LibraryPersistentBean ejbName:
LibraryPersistentBean
```

```
10:51:37,287 INFO  [JndiSessionRegistrarBase] Binding the following Entries in
Global JNDI:


     LibraryPersistentBean/remote - EJB3.x Default Remote Business Interface

     LibraryPersistentBean/remote-
com.tutorialspoint.stateless.LibraryPersistentBeanRemote - EJB3.x Remote
Business Interface


10:51:37,349 INFO  [EJBContainer] STARTED EJB:
com.tuturialspoint.messagebean.LibraryMessageBean ejbName: BookMessageHandler

10:51:37,443 INFO  [DefaultEndpointRegistry] register:
jboss.ws:context=EjbComponent,endpoint=LibraryPersistentBean

10:51:38,191 INFO  [WSDLFilePublisher] WSDL published to: file:/D:/Jboss-
5.0.1/server/default/data/wsdl/EjbComponent.jar/

LibraryService3853081455302946642.wsdl
```

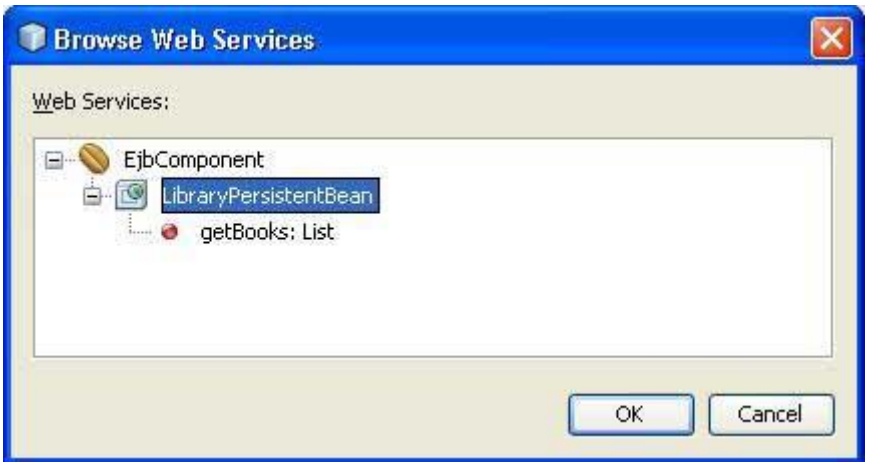
Create Client to Access EJB as Web Service

In NetBeans IDE, select, **File > New Project >**.Select project type under category, **Java**, Project type as **Java Application**. Click **Next >** button.Enter project name and location. Click **Finish >** button. We have chosen name as EJBWebServiceClient.

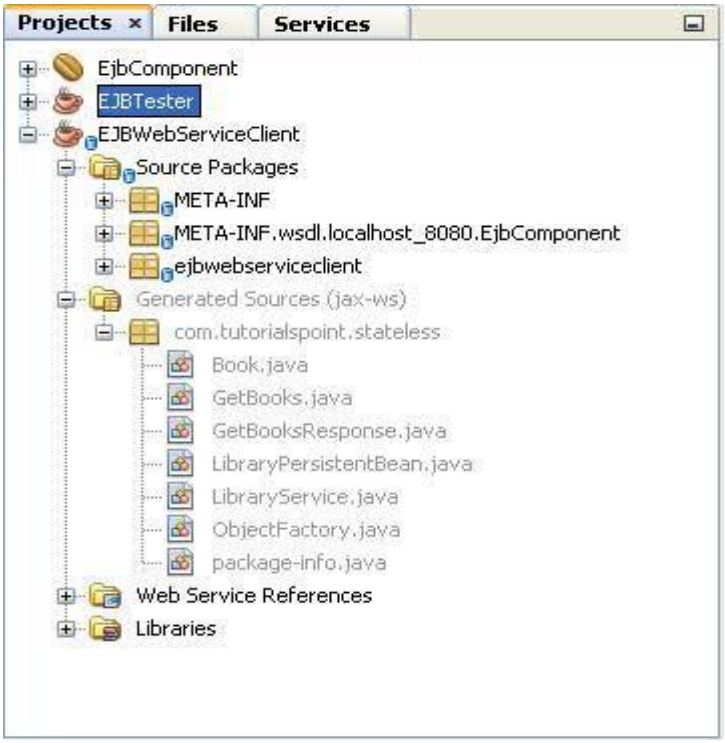
Right click on the project name in Project explorer window. Select **New > WebService Client**.



Add EJB component project's LibraryPersistentBean created earlier under WSDL and Client Location using **Add Project** button in **compile** tab.



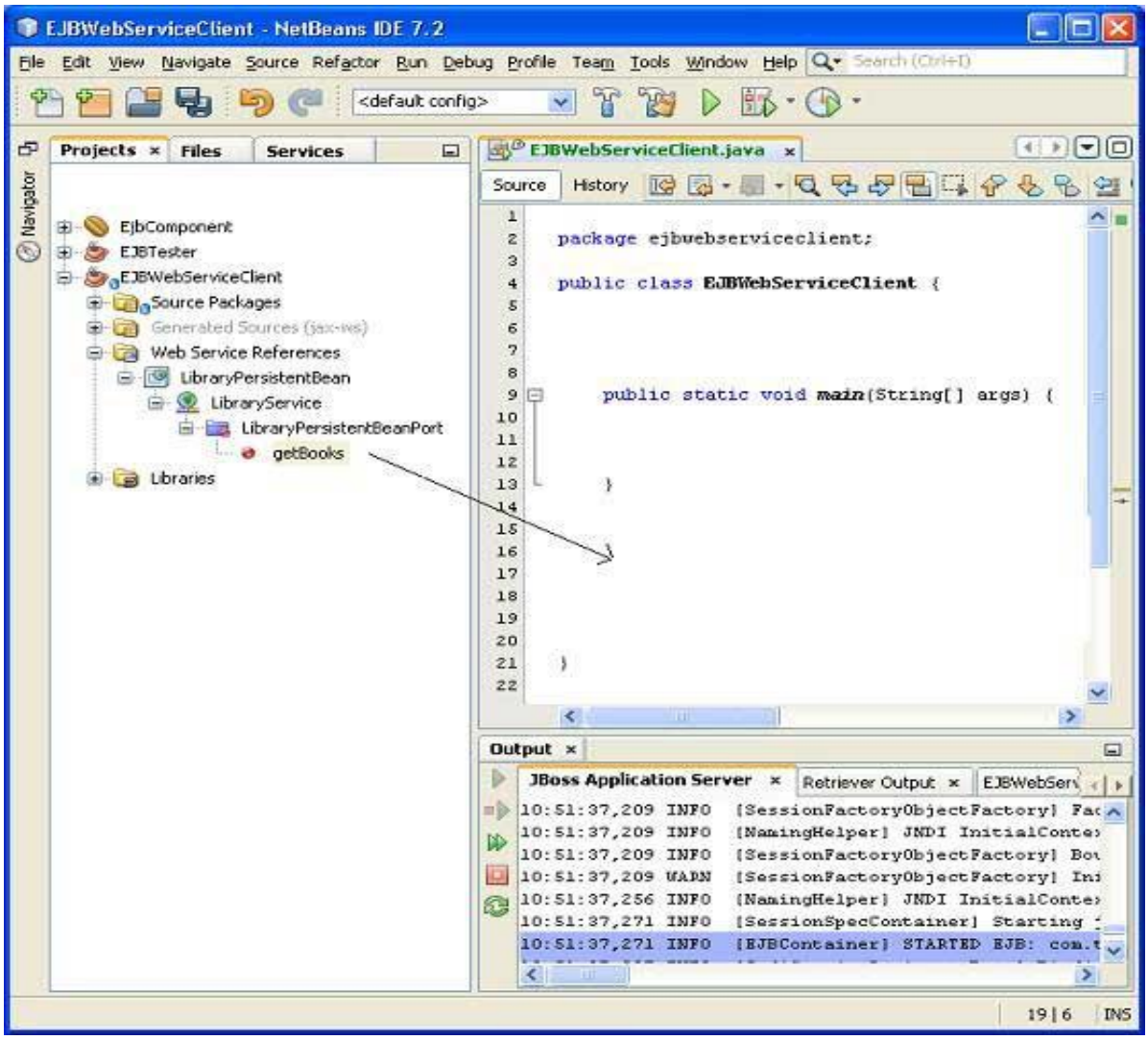
Click Finish Button. Verify the following structure in project explorer.



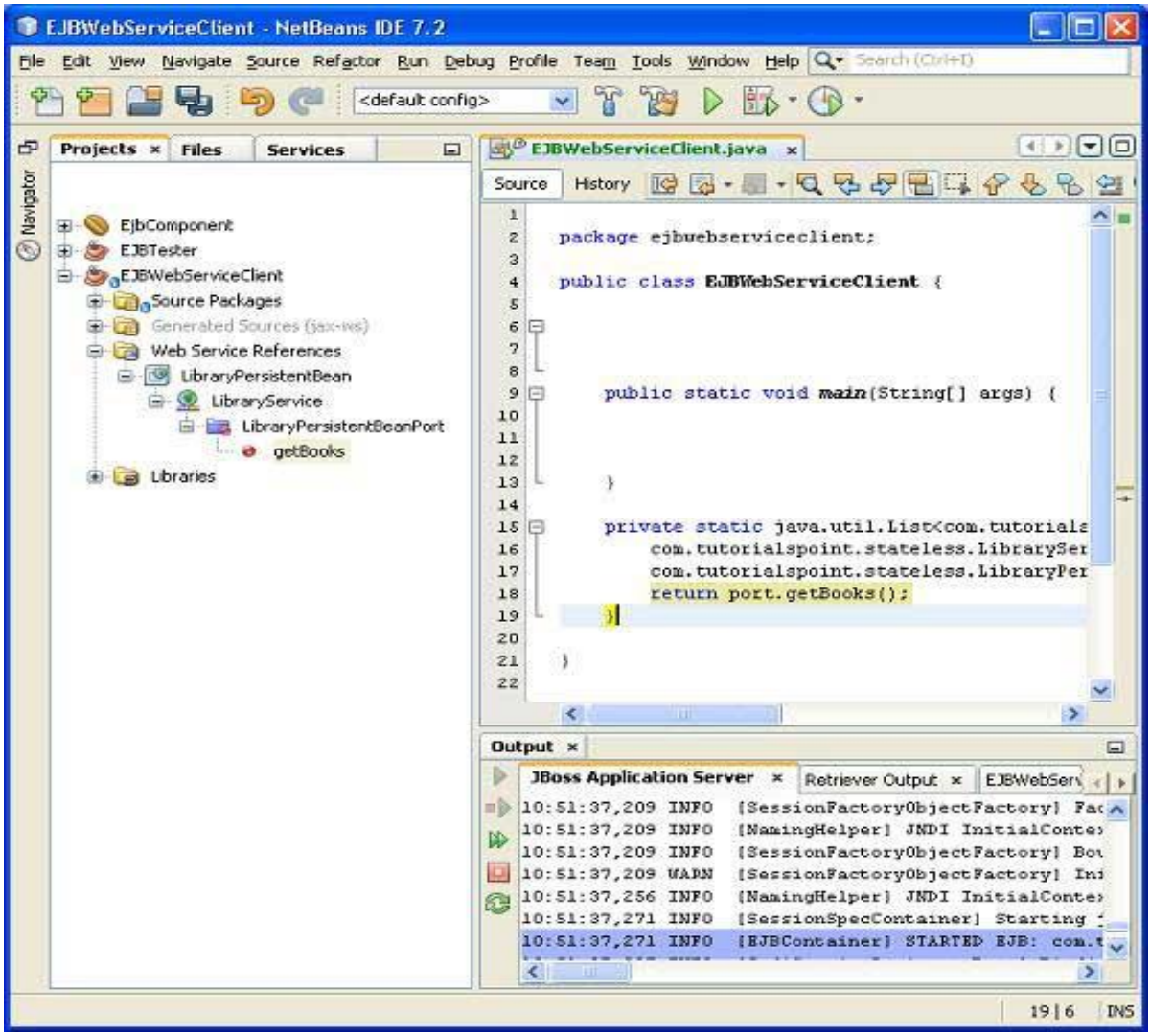
Create EJBWebServiceClient.java

```
package ejbwebserviceclient;


public class EJBWebServiceClient {
   public static void main(String[] args) {
   }
}
```

Select Web Service getBooks web method as shown in the figure below and drag it to code window of EJBWebServiceClient.



You will see the output similar to as shown below.



Update the EJBWebServiceClient code to use this method.

```
package ejbwebserviceclient;

public class EJBWebServiceClient {

   public static void main(String[] args) {
      for(com.tutorialspoint.stateless.Book book:getBooks()){
         System.out.println(book.getName());
      }
   }

   private static java.util.List
   <com.tutorialspoint.stateless.Book> getBooks() {
```

```
      com.tutorialspoint.stateless.LibraryService service =
         new com.tutorialspoint.stateless.LibraryService();
      com.tutorialspoint.stateless.LibraryPersistentBean port =
         service.getLibraryPersistentBeanPort();
      return port.getBooks();
   }
}
```

Run the Client

Right click on the project name in Project explorer window. Select **Run**. Netbeans will build the client and run it. Verify the following output.

```
ant -f D:\\SVN\\EJBWebServiceClient run
init:
Deleting: D:\SVN\EJBWebServiceClient\build\built-jar.properties
deps-jar:
Updating property file: D:\SVN\EJBWebServiceClient\build\built-jar.properties
wsimport-init:
wsimport-client-LibraryPersistentBean:
files are up to date
classLoader = java.net.URLClassLoader@4ce46c
SharedSecrets.getJavaNetAccess()=java.net.URLClassLoader$7@182cdac
wsimport-client-generate:
Compiling 1 source file to D:\SVN\EJBWebServiceClient\build\classes
compile:
run:
learn java
Learn Spring
learn JSF
Learn HTML
Learn JBoss
Learn EJB
Learn Hibernate
Learn IBatis
Times Now
learn html5
```

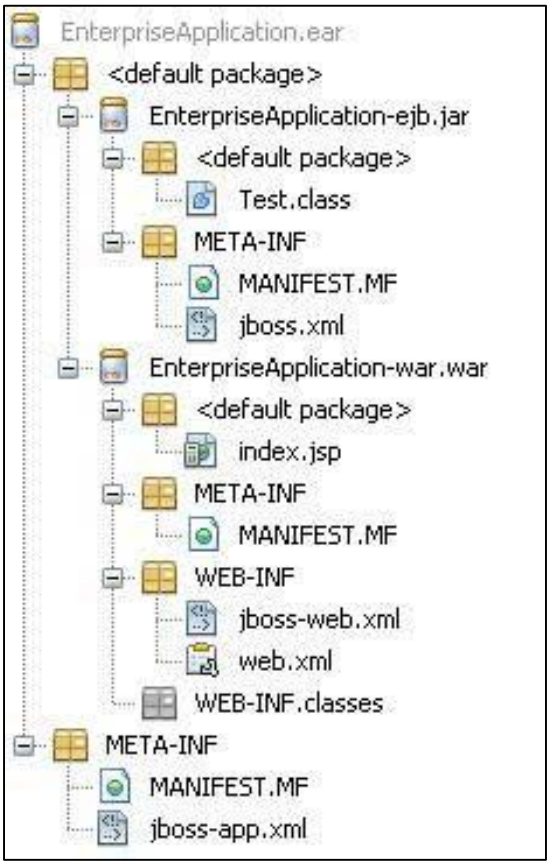
```
Learn images

Learn Testing

Forbes
test1
BUILD SUCCESSFUL (total time: 1 second)
```

23. EJB – PACKAGING APPLICATIONS

Requirement of Packaging applications using EJB 3.0 are similar to that of J2EE platform. EJB components are packaged into modules as jar files and are packaged into application enterprise archive as ear file.

There are majorly three components of any enterprise application:

- **jar** - Java Application aRchive, containing EJB modules, EJB client modules and utility modules.
- **war** - Web Application aRchive, containing web modules.
- **ear** - Enterprise Application aRchive, containing jars and war module.



In NetBeans, it is very easy to create, develop, package, and deploy the J2EE applications.

In NetBeans IDE, select, **File > New Project >**.Select project type under category, **Java EE**, Project type as **Enterprise Application**. Click **Next >** button. Enter project name and location. Click **Finish >** button. We've chosen name as EnterpriseApplicaton.

Select Server and Settings. Keep **Create EJB Module** and **Create Web Application Module** checked with default names provided. Click finish button. NetBeans will create the following structure in project window.

Right click on the Project **Enterprise Application** in project explorer and select Build.

```
ant -f D:\\SVN\\EnterpriseApplication dist
pre-init:
init-private:
init-userdir:
init-user:
init-project:
do-init:
post-init:
init-check:
init:
deps-jar:
deps-j2ee-archive:
EnterpriseApplication-ejb.init:
EnterpriseApplication-ejb.deps-jar:
EnterpriseApplication-ejb.compile:
EnterpriseApplication-ejb.library-inclusion-in-manifest:
```

```
Building jar: D:\SVN\EnterpriseApplication\EnterpriseApplication-
ejb\dist\EnterpriseApplication-ejb.jar

EnterpriseApplication-ejb.dist-ear:
EnterpriseApplication-war.init:
EnterpriseApplication-war.deps-module-jar:
EnterpriseApplication-war.deps-ear-jar:
EnterpriseApplication-ejb.init:
EnterpriseApplication-ejb.deps-jar:
EnterpriseApplication-ejb.compile:
EnterpriseApplication-ejb.library-inclusion-in-manifest:
EnterpriseApplication-ejb.dist-ear:
EnterpriseApplication-war.deps-jar:
EnterpriseApplication-war.library-inclusion-in-archive:
EnterpriseApplication-war.library-inclusion-in-manifest:
EnterpriseApplication-war.compile:
EnterpriseApplication-war.compile-jsps:
EnterpriseApplication-war.do-ear-dist:

Building jar: D:\SVN\EnterpriseApplication\EnterpriseApplication-
war\dist\EnterpriseApplication-war.war

EnterpriseApplication-war.dist-ear:
pre-pre-compile:
pre-compile:
Copying 1 file to D:\SVN\EnterpriseApplication\build
Copying 1 file to D:\SVN\EnterpriseApplication\build
do-compile:
post-compile:
compile:
pre-dist:
do-dist-without-manifest:
do-dist-with-manifest:

Building jar: D:\SVN\EnterpriseApplication\dist\EnterpriseApplication.ear
```

```
post-dist:

dist:

BUILD SUCCESSFUL (total time: 1 second)
```

Here you can see that Netbeans prepares Jar file first, then War file, and in the end, the ear file, carrying the jar and war file. Each jar, war, and ear file carries a **meta-inf** folder to have meta-data as per the J2EE specification.