How to create EJB3 JPA Project with JAX-RS In Eclipse (Jboss AS 7.1)

By Bikash Mohanty

18th February 2020

ENVIRONMENT USED

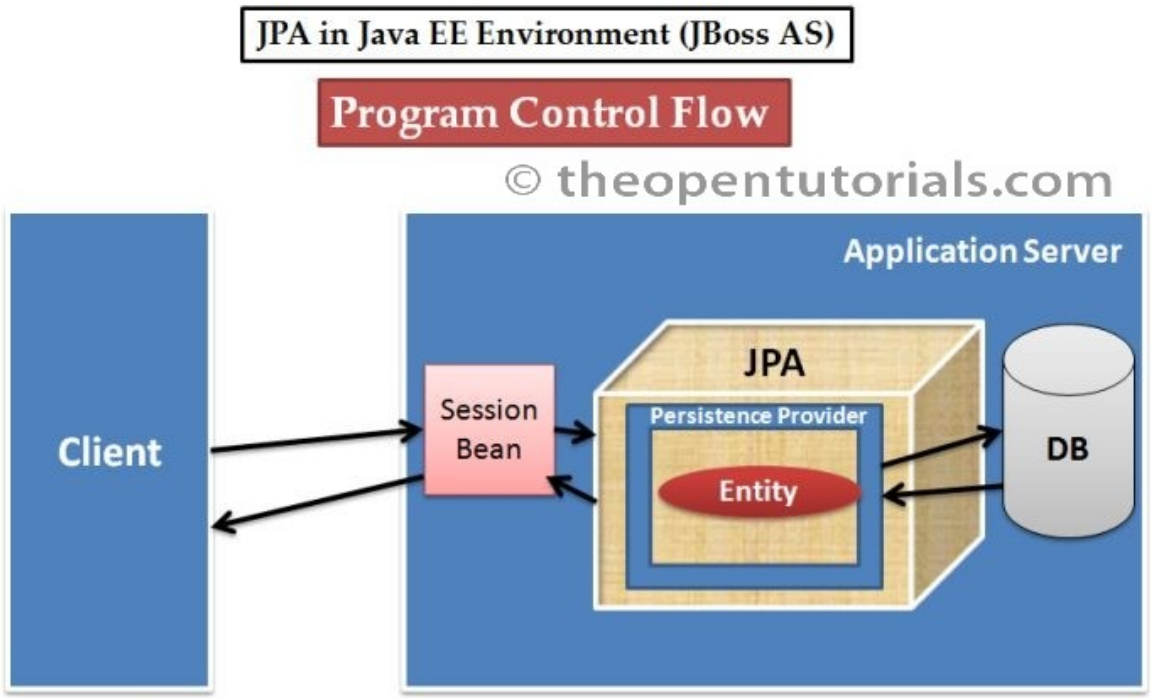
- JDK 6 (Java SE 6)
- EJB 3.0 (stateless session bean)
- EJB 3.0 Java Persistence API (JPA)
- Eclipse Indigo IDE for Java EE Developers (3.7.1)
- JBoss Tools – Core 3.3.0 M5 for Eclipse Indigo (3.7.1)
- JBoss Application Server (AS) 7.1.0.CR1b / Final
- MySQL 5.5 (To install MySQL refer this page)
- MySQL Connector/J 5.1

SETTING UP DEVELOPMENT ENVIRONMENT:

Read this page for installing and setting up the environment for developing and deploying EJB 3.0 on JBoss application server.

PROJECT DESCRIPTION:

- We are going to create a simple EJB 3 JPA project and a remote Java application client which will call/invoke the bean.
- We create a JPA entity and a stateless session bean to perform operations on the entity.
- For testing this JPA example we write a remote Java Application Client (main() method).
- For simplicity, the entity, session bean and the client are created in the same project.



STEPS

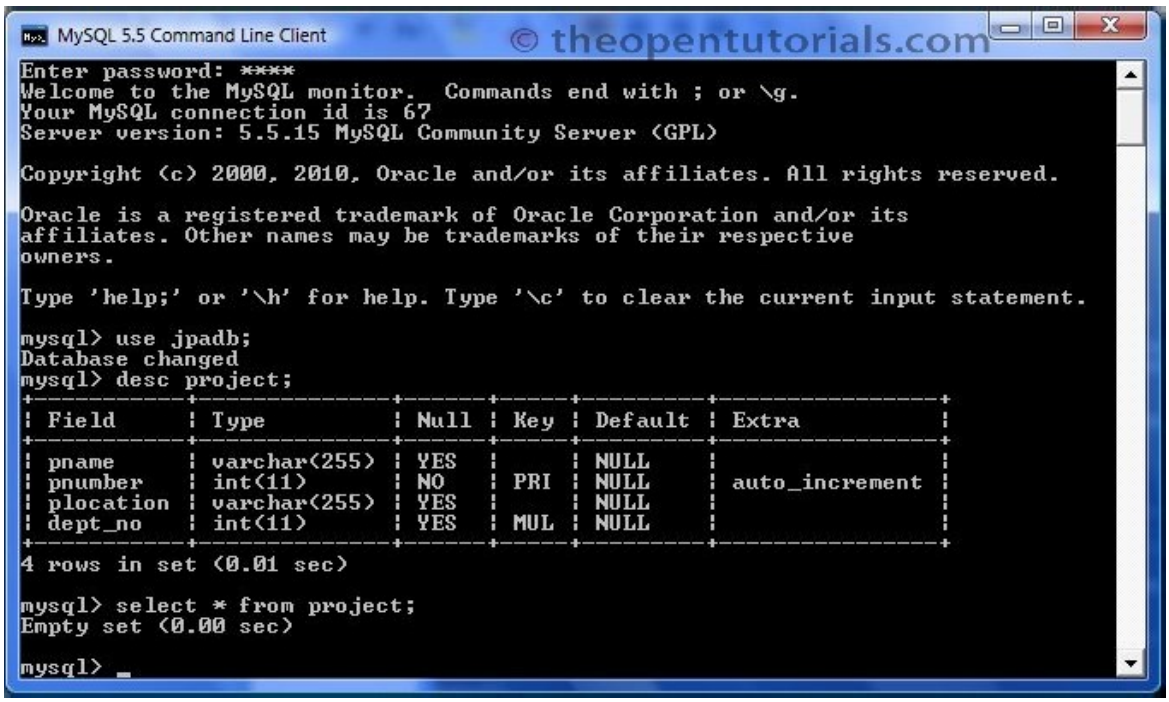
1. Create Database Table
2. Create JPA Entity
    - POJO class with @Entity annotation
    - persistence.xml
    - [optional] orm.xml if object relational mapping is de﷿ned in XML
3. Create Stateless Session Bean
    - Bean interface
    - Bean Implementation class
4. Create Client
    - Client Class with main() method
    - jboss-ejb-client.properties for de﷿ning JBoss speci﷿c client context in JBoss AS7
    - JAR ﷿les for accessing Session Bean
    - MySQL connector JAR ﷿le
5. Adding MySQL data source in JBoss AS

CREATING DATABASE AND TABLE IN MYSQL

JPA is all about data persistence, so let’s examine how it works with the data store design. Assume you have a PROJECT table, as shown below.

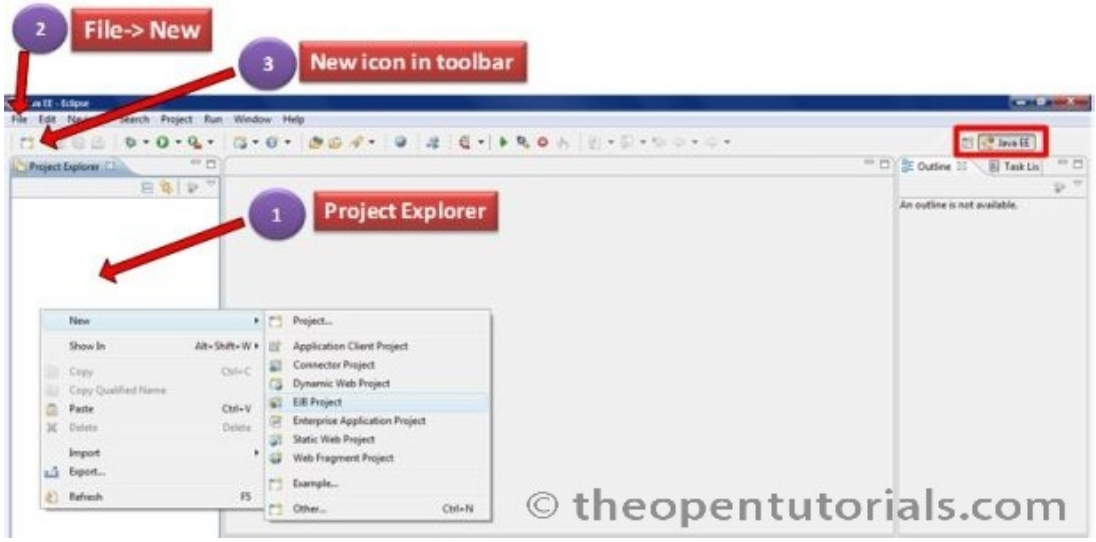
FIELD	TYPE	KEY	EXTRA
pname	varchar(255)		
pnumber	int	Primary Key	auto_increment
plocation	varchar(255)		
dept_no			

- Open command prompt (Windows) or Terminal(Linux) and type
  mysql –u [your-username] –p
  and press enter and type the password.
- If you are using Windows, you can also use MySQL command line client which will be available in All programs menu.
- For creating a new database, refer this page.
- After creating the database type the command “use <database_name>;”
- For creating a new table, refer this page.

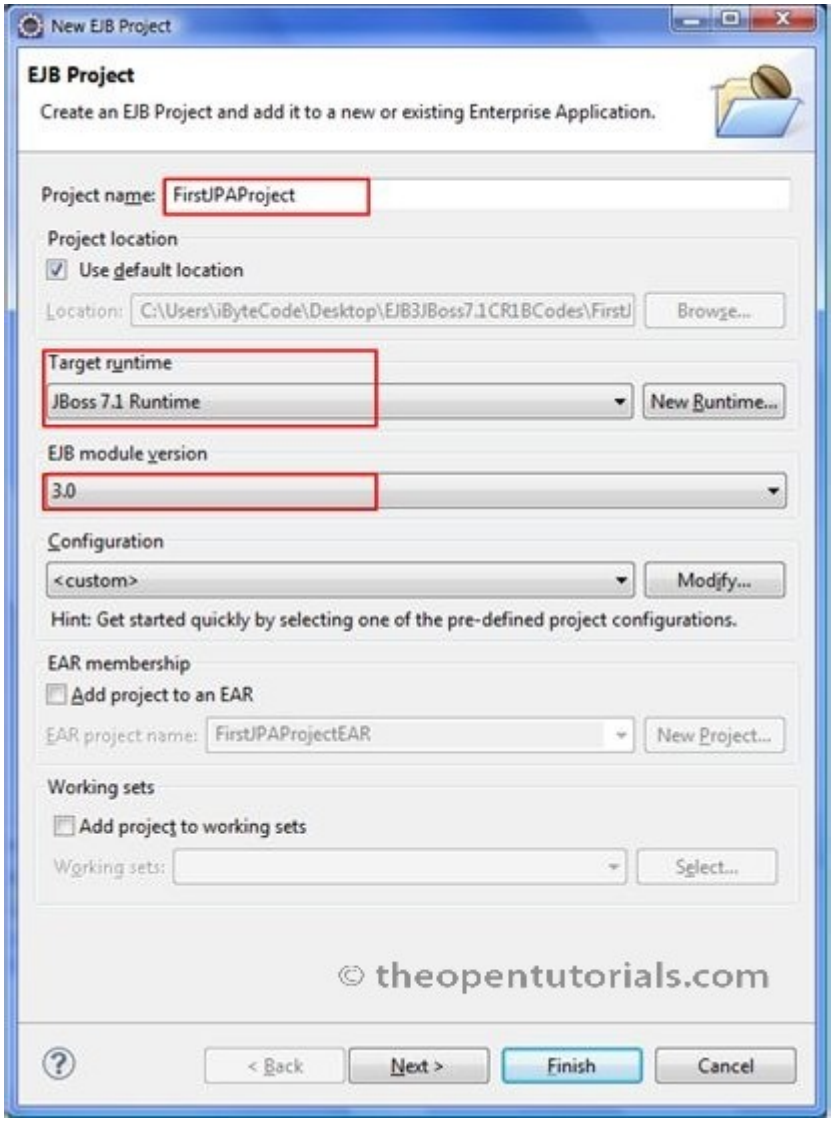


CREATING NEW EJB PROJECT

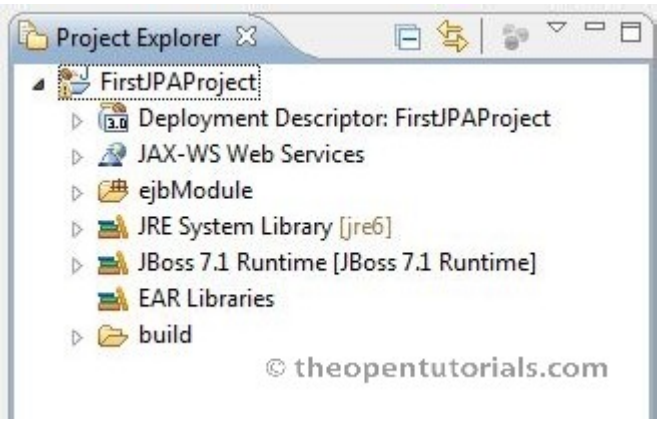
- Open Eclipse IDE and create a new EJB project which can be done in three ways,
  - Right click on Project Explorer -> New -> EJB Project
  - File menu -> New -> EJB Project
  - Click on the down arrow on New icon on toolbar -> EJB Project



- Enter the project name as “FirstJPAProject” and make sure the JBoss 7.1 Runtime has been selected with the EJB 3.0 Module version.



- Click Next -> Next -> and Finish.
- You will see an EJB project in the Project Explorer view.



CREATING JPA ENTITY

This is a very simple example that uses only one entity – “Project” which is a Plain Old Java Object class (POJO). This class, as well as the code that manipulates POJO instances, can be used without any changes in Java SE or Java EE environment. In this example we have used Java EE environment.

We will persist and find “project” entity using EntityManager API and retrieve all “projects” using Query interface.

Right click on ejbModule -> New -> Class

- Enter the Java package name as “com.ibytecode.entities”
- Enter the Class name as “Project”
- Click “Finish”

Type the following code:

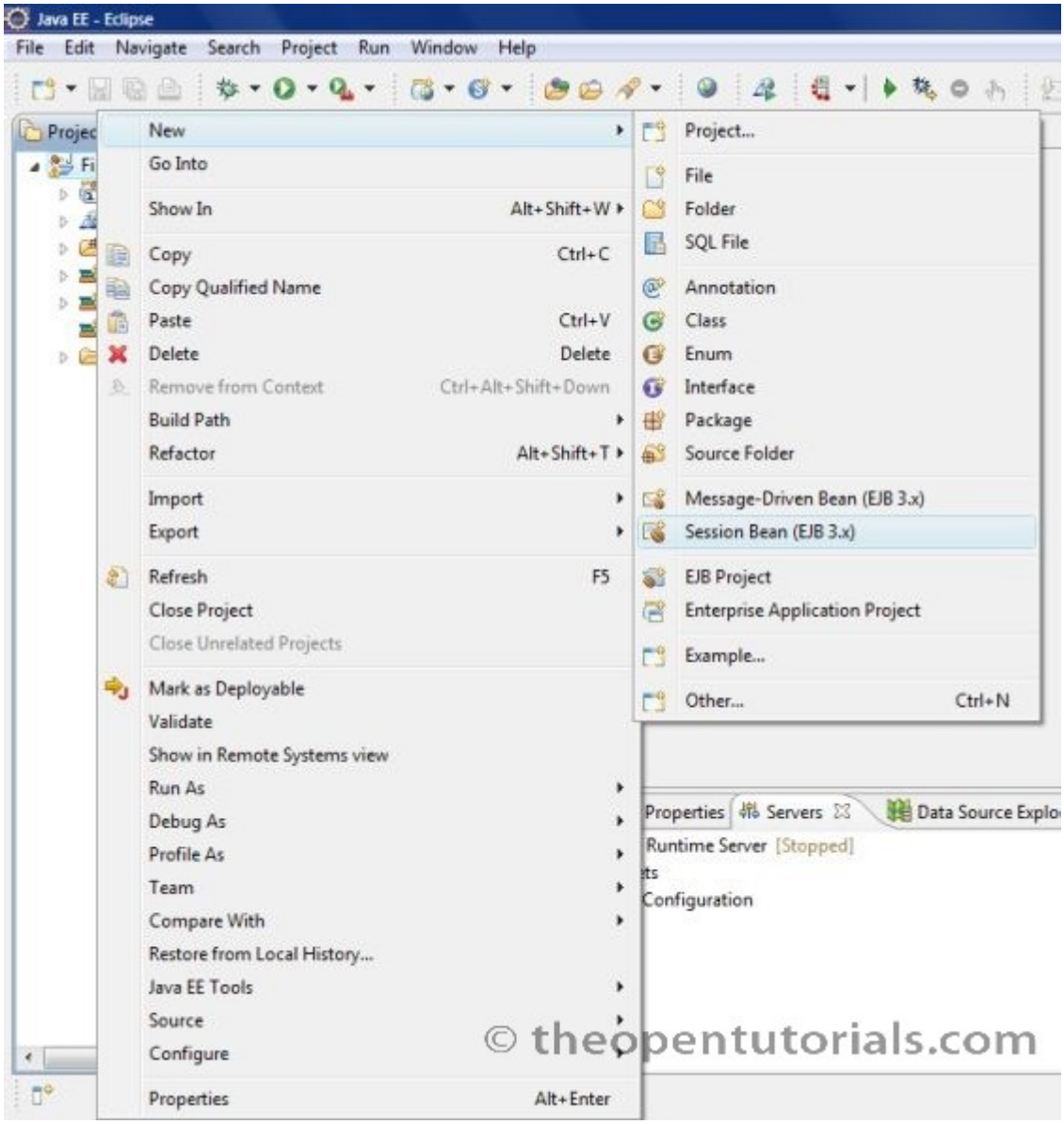
```
package com.ibytecode.entities;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Column;

@Entity(name = "project")
public class Project implements Serializable {
        private static final long serialVersionUID = 1L;

        public Project() {
                super();
        }

        @Id
        private int pnumber;
        private String pname;
        private String plocation;

        @Column(name = "dept_no")
        private int deptNo;
```

```
	public int getPnumber() {
		return pnumber;
	}
	public void setPnumber(int pnumber) {
		this.pnumber = pnumber;
	}
	public String getPname() {
		return pname;
	}
	public void setPname(String pname) {
		this.pname = pname;
	}
	public String getPlocation() {
		return plocation;
	}
	public void setPlocation(String plocation) {
		this.plocation = plocation;
	}
	public int getDeptNo() {
		return deptNo;
	}
	public void setDeptNo(int deptNo) {
		this.deptNo = deptNo;
	}
	@Override
	public String toString() {
		return "Project [pnumber=" + pnumber + ", pname=" + pname
				+ ", plocation=" + plocation + ", deptNo=" +
deptNo + "]";
	}
}
```

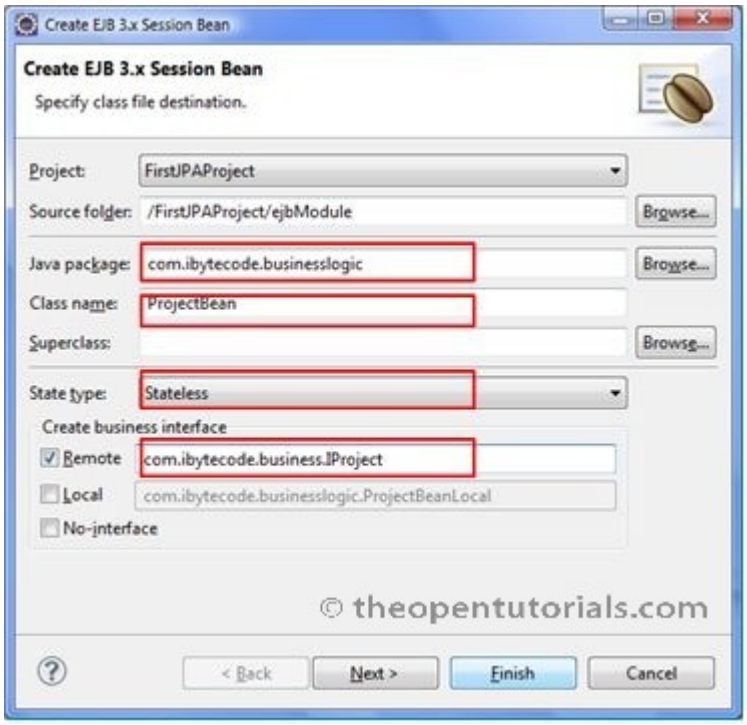
Note that there is no @Table annotation. This is possible because the persistence provider will use the default rules to calculate the values for you. The name attribute in @Entity annotation deﬁnes the table name. Similarly if an instance variable name matches the column name in the table then there is no need to specify the @Column annotation.

CREATING SESSION BEAN AND BEAN INTERFACE

- Right click on ejbModule -> New -> Session Bean (EJB 3.x)



- Enter the Java package name as com.ibytecode.businesslogic
- Enter the Class name as ProjectBean
- Select the State type as Stateless
- Check the Remote Business Interface and enter the name as com.ibytecode.business.IProject.
- The business interface will be created in different package (com.ibytecode.business)
- Click Finish



CODING BEAN AND THE INTERFACE

- Open Bean Interface and type the following code and save the file (Ctrl+s).
- Interface can be either @Remote or @Local. In this example we have used @Remote.

```
package com.ibytecode.business;
import java.util.List;
import javax.ejb.Remote;

import com.ibytecode.entities.Project;

@Remote
public interface IProject {
        void saveProject(Project project);
        Project findProject(Project project);
        List<Project> retrieveAllProjects();
}
```

- Open Bean and type the following code and save the file.
- Bean type can either be @Stateful or @Stateless. In this example we have used @Stateless.

```
package com.ibytecode.businesslogic;

import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;

import com.ibytecode.business.IProject;
import com.ibytecode.entities.Project;

@Stateless
public class ProjectBean implements IProject {

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

    public ProjectBean() {   }

        @Override
        public void saveProject(Project project) {
                entityManager.persist(project);
        }

        @Override
        public Project findProject(Project project) {
                Project p = entityManager.find(Project.class,
project.getPnumber());
                return p;
        }

        @Override
        public List<Project> retrieveAllProjects() {

                String q = "SELECT p from " + Project.class.getName() + " p";
                Query query = entityManager.createQuery(q);
                List<Project> projects = query.getResultList();
                return projects;
        }
}
```

Now the Stateless Session Bean has been created. The next step is to con�gure the datasource.

PERSISTENCE.XML

How does the server know which database the EntityManager API should use to save / update / query the entity objects? The persistence.xml file gives you complete flexibility to configure the EntityManager.

The persistence.xml file is a standard configuration file in JPA which should be placed in META-INF directory inside the JAR file that contains the entities. The persistence.xml file must define a persistence-unit with a unique name which is used by EntityManager.

Right click on META-INF folder -> New -> Other -> XML -> XML file. Enter the file name as persistence.xml and type the following.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" 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_2_0.xsd">
        <persistence-unit name="JPADB">
        <jta-data-source>java:/MySQLDS</jta-data-source>
                <properties>
                        <property name="showSql" value="true"/>
                        <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQLDialect" />
                </properties>
        </persistence-unit>
</persistence>
```

In JBoss AS, the default JPA provider is Hibernate. The jta-data-source points to the JNDI name of the database this persistence unit maps to. The java:/MySQLDS points to the MySQL DB datasource in the JBoss AS. In the next step we setup this datasource.

CONFIGURING MYSQL DATASOURCE IN JBOSS AS 7

DOWNLOAD MYSQL CONNECTOR

The connector can be downloaded from this link. This tutorial uses 5.1 version. Unzip the connector to a safe location on your computer which contains MySQL Connector J JAR.

ADD A MODULE TO AS 7

AS 7 uses a module system to provide isolation in class loading. We need to create a new module which contains the MySQL Connector J JAR.

In your JBoss AS 7 root folder, create folders in following hierarchy, modules/com/mysql/main.

If “modules” and “com” folders are already present then just create “mysql” and “main” folders.
Copy the MySQL Connector J JAR and paste in the “main” folder.

Now define the module in XML. Create a new module.xml file in “main” folder and paste the following lines.

```
<?xml version="1.0" encoding="UTF-8"?>

<module xmlns="urn:jboss:module:1.0" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.18-bin.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
  </dependencies>
</module>
```

The new module directory should have the following contents.

- module.xml
- mysql-connector-java-5.1.18-bin.jar

CREATE A DRIVER REFERENCE

Now we need to make a reference to the module from the main application server conﬁguration ﬁle (standalone.xml) which is found in JBossAS_Home/standalone/conﬁguration

Find the ‘<drivers>’ element and add a new driver to it:

```
<drivers>
        <driver name="h2" module="com.h2database.h2">
                <xa-datasource-class>
                        org.h2.jdbcx.JdbcDataSource
                </xa-datasource-class>
        </driver>
        <driver name="mysqlDriver" module="com.mysql">
                <xa-datasource-class>
                        com.mysql.jdbc.Driver
                </xa-datasource-class>
        </driver>
</drivers>
```

ADD THE DATASOURCE FOR THE DRIVER

Open the application server conﬁguration ﬁle (standalone.xml) which is found in JBossAS_Home/standalone/conﬁguration. Find the ‘<datasources>’ element and add a new datasource.

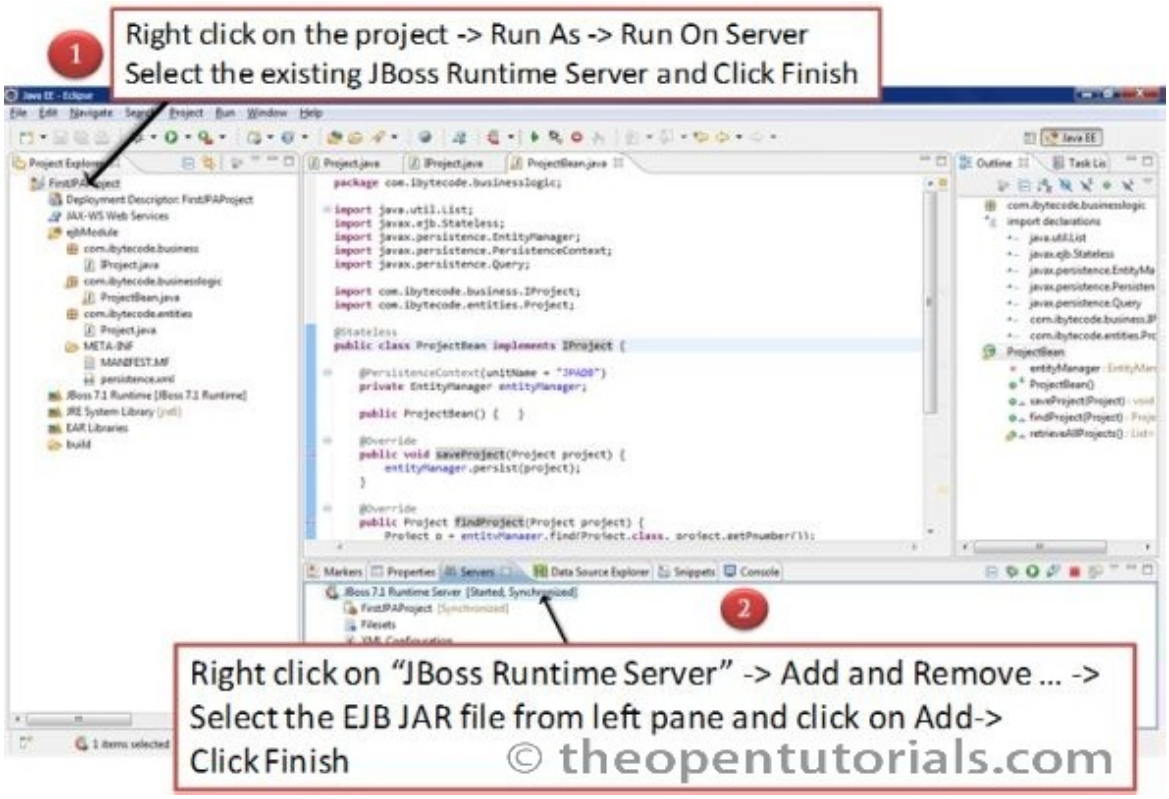
```
<datasource jndi-name="java:/MySQLDS"
        pool-name="MySQLDS" enabled="true" use-java-context="true">
        <connection-url>
                jdbc:mysql://localhost:3306/YOUR-DATABASE-NAME
        </connection-url>
        <driver>mysqlDriver</driver>
        <security>
                <user-name>YOUR-MYSQL-USERNAME</user-name>
                <password>YOUR-MYSQL-PASSWORD</password>
        </security>
</datasource>
```

In the above code, use your database name, MySQL username and password in the highlighted lines.

In datasource element, the jndi-name=”java:/MySQLDS” should match the java:/MySQLDS in persistence.xml.
The value for <driver>mysqlDriver</driver> element should match the <drivers><driver name=”mysqlDriver” …>…</driver></drivers>

DEPLOYING EJB JPA PROJECT

- Now we need to deploy the project “FirstJPAProject” on server.
- Deploying the project can be done in two ways,
  - Right click on the EJB project -> Run As -> Run On Server. Select the existing “JBoss 7.1 Runtime Server” and click Finish.
  - Right click on “JBoss 7.1 Runtime Server” available in Servers view -> Add and Remove… -> Select the EJB JAR file from the left pane and click Add-> and then Finish.



START/RESTART THE SERVER

Right click on “JBoss 7.1 Runtime Server” from Servers view and click on Start if it has not yet been started. If the project is deployed properly with global JNDI mapping then you will see the following message in the console.

Deployed “FirstJPAProject.jar”

CREATING CLIENT

- The next step is to write a remote Java client application (with main()) for accessing and invoking the bean deployed on the server
- Client uses JNDI to lookup for a proxy of your bean and invokes method on that proxy.

CREATING JNDI INITIALCONTEXT

Obtaining a Context using InitialContext

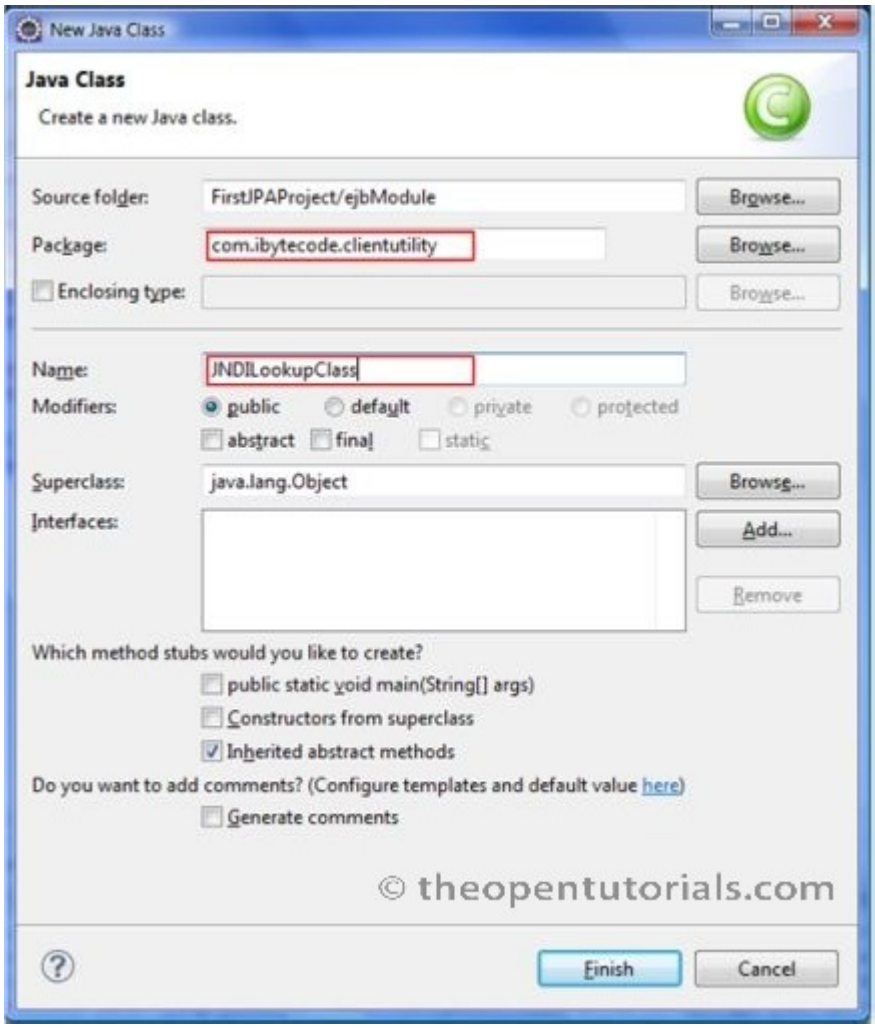
- All naming service operations are performed on some implementation of the javax.naming.Context interface. Therefore, the starting point of interacting with the naming service is to obtain a Context by providing the properties speci�c to the server implementation being used. In our case it is, JBoss Application Server.
- To create a javax.naming.InitialContext, we need to initialize it with properties from the environment. JNDI veri�es each property’s value by merging the values from the following two sources,
  - Using parameterized constructor of InitialContext which takes properties of supplied environment
  - jndi.properties resource �les found on the classpath.

NOTE:We will use parameterized constructor for initializing the InitialContext.

For JBoss AS 7 we need to set the Context.URL_PKG_PREFIXES property with value “org.jboss.ejb.client.naming” to obtain the InitialContext.

The following utility class is used to create InitialContext for JBoss AS and can be reused in all applications. Otherwise the code written in this class should be repeated in all clients.

- Right click on ejbModule -> New -> Class
- Enter the package name as com.ibytecode.clientutility
- Enter the Class name as JNDILookupClass
- Click on Finish



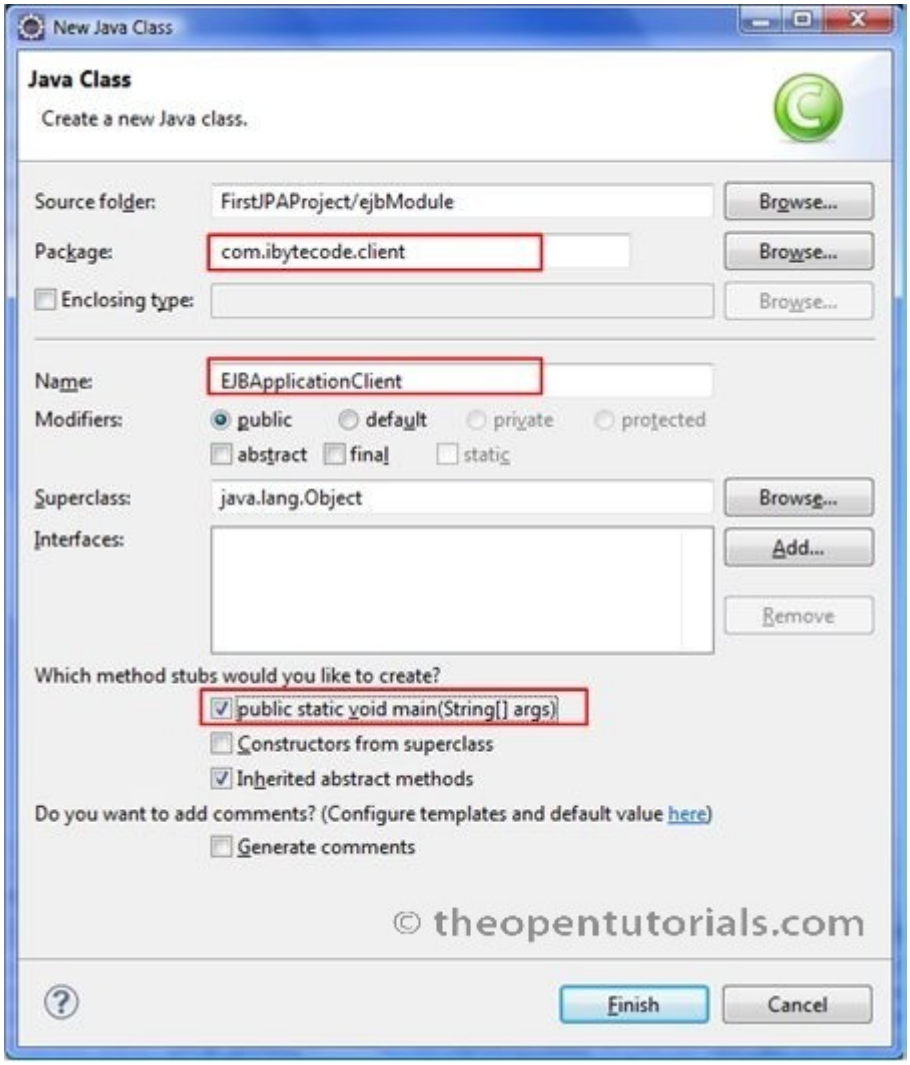
Type the following code.

```
package com.ibytecode.clientutility;

import java.util.Properties;
import javax.naming.Context;
import javax.naming.InitialContext;
```

```
import javax.naming.NamingException;

public class JNDILookupClass {

        private static Context initialContext;

        private static final String PKG_INTERFACES =
"org.jboss.ejb.client.naming";

        public static Context getInitialContext() throws NamingException {
                if (initialContext == null) {
                        Properties properties = new Properties();
                        properties.put(Context.URL_PKG_PREFIXES, PKG_INTERFACES);

                        initialContext = new InitialContext(properties);
                }
                return initialContext;
        }
}
```

CREATING CLIENT CLASS

- Right click on ejbModule -> New -> Class
- Enter the package name as com.ibytecode.client
- Enter the Class name as EJBApplicationClient
- Check the main() method option
- Click on Finish



Type the following code:

```
package com.ibytecode.client;

import java.util.List;

import javax.naming.Context;
import javax.naming.NamingException;

import com.ibytecode.business.IProject;
import com.ibytecode.businesslogic.ProjectBean;
import com.ibytecode.clientutility.JNDILookupClass;
import com.ibytecode.entities.Project;

public class EJBApplicationClient {
```

```
 public static void main(String[] args) {
        IProject bean = doLookup();

        Project p1 = new Project();
        p1.setPname("Banking App");
        p1.setPlocation("Town City");
        p1.setDeptNo(1);

        Project p2 = new Project();
        p2.setPname("Office Automation");
        p2.setPlocation("Downtown");
        p2.setDeptNo(2);

        // 4. Call business logic
        //Saving new Projects
        bean.saveProject(p1);
        bean.saveProject(p2);

        //Find a Project
        p1.setPnumber(1);
        Project p3 = bean.findProject(p1);
        System.out.println(p3);

        //Retrieve all projects
System.out.println("List of Projects:");
        List<Project> projects = bean.retrieveAllProjects();
        for(Project project : projects)
                System.out.println(project);


 }

 private static IProject doLookup() {
        Context context = null;
        IProject bean = null;
        try {
                // 1. Obtaining Context
                context = JNDILookupClass.getInitialContext();
                // 2. Generate JNDI Lookup name
                String lookupName = getLookupName();
                // 3. Lookup and cast
                bean = (IProject) context.lookup(lookupName);

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

```
                return bean;
        }

        private static String getLookupName() {
                /*The app name is the EAR name of the deployed EJB without .ear
                suffix. Since we haven't deployed the application as a .ear, the
app
                name for us will be an empty string */
                String appName = "";

                /* The module name is the JAR name of the deployed EJB without
the
                .jar suffix.*/
                String moduleName = "FirstJPAProject";

                /* AS7 allows each deployment to have an (optional) distinct
name.
                This can be an empty string if distinct name is not specified.*/
                String distinctName = "";

                // The EJB bean implementation class name
                String beanName = ProjectBean.class.getSimpleName();

                // Fully qualified remote interface name
                final String interfaceName = IProject.class.getName();

                // Create a look up string name
                String name = "ejb:" + appName + "/" + moduleName + "/" +
                                distinctName    + "/" + beanName + "!" +
interfaceName;
                return name;
        }
}
```

SETTING UP EJB CLIENT CONTEXT PROPERTIES

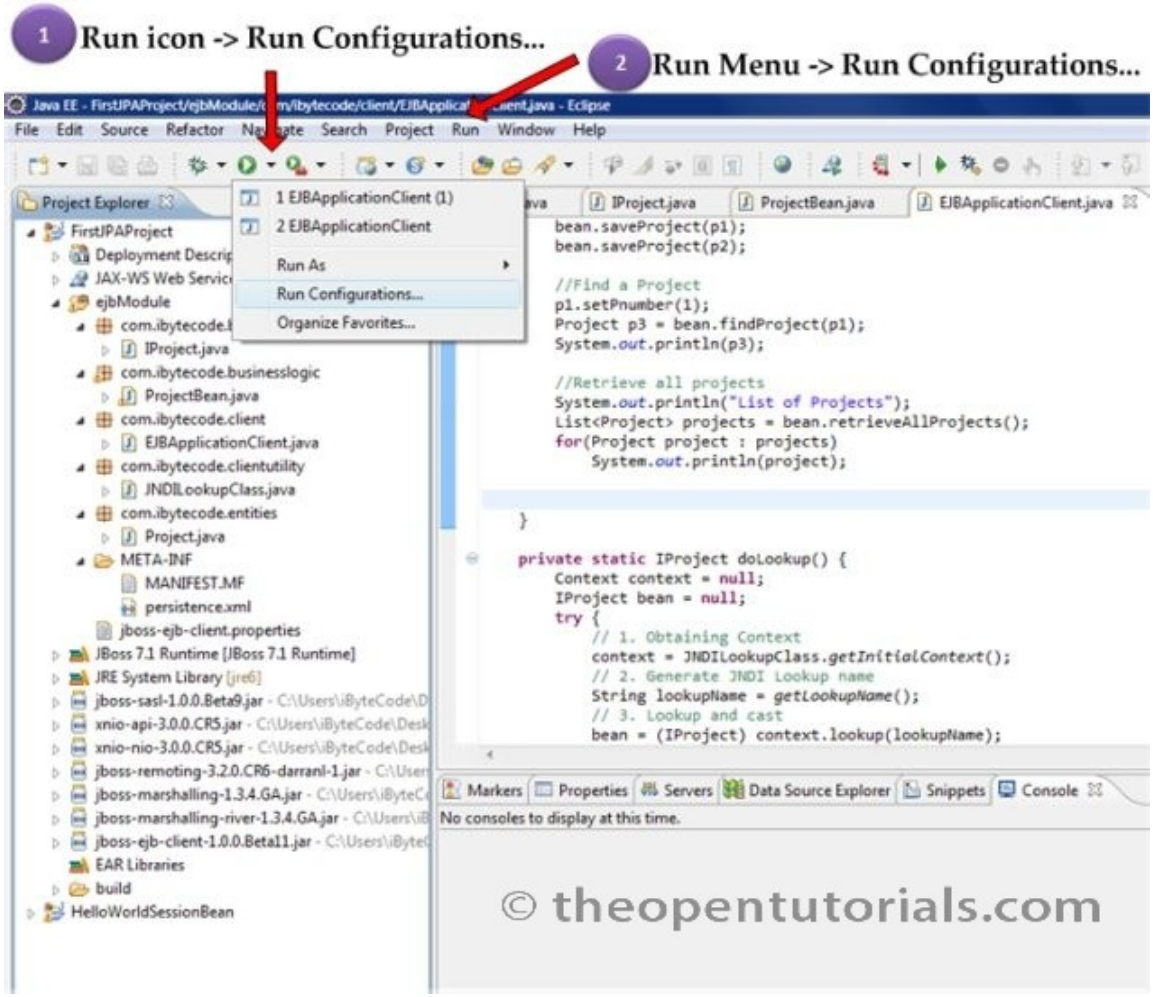
An EJB client context is a context which contains contextual information for carrying out remote invocations on EJBs. This is a JBoss AS speciﬁc API. The EJB client context can be associated with multiple EJB receivers. Each EJB receiver is capable of handling invocations on different EJBs. For example, an EJB receiver “ClientA” might be able to handle invocation on a bean identiﬁed by app-A/module-A/distinctinctName-A/BeanA!com.ibc.RemoteBeanA, app-B/module-B/distinctName-B/BeanB!RemoteBeanB, etc. Each such EJB receiver knows about what set of EJBs it can handle and each of the EJB receiver knows which server target to use for handling the invocations on the bean. The server IP address and its remoting port should be speciﬁed in the properties ﬁle placed in the client classpath. This properties ﬁle (EJB client context) will then be used internally by the JNDI implementation to handle invocations on the bean proxy.

Create a ﬁle “jboss-ejb-client.properties” in the classpath of the application. We can place it in ejbModule folder of our application. The jboss-ejb-client.properties contains the following properties:

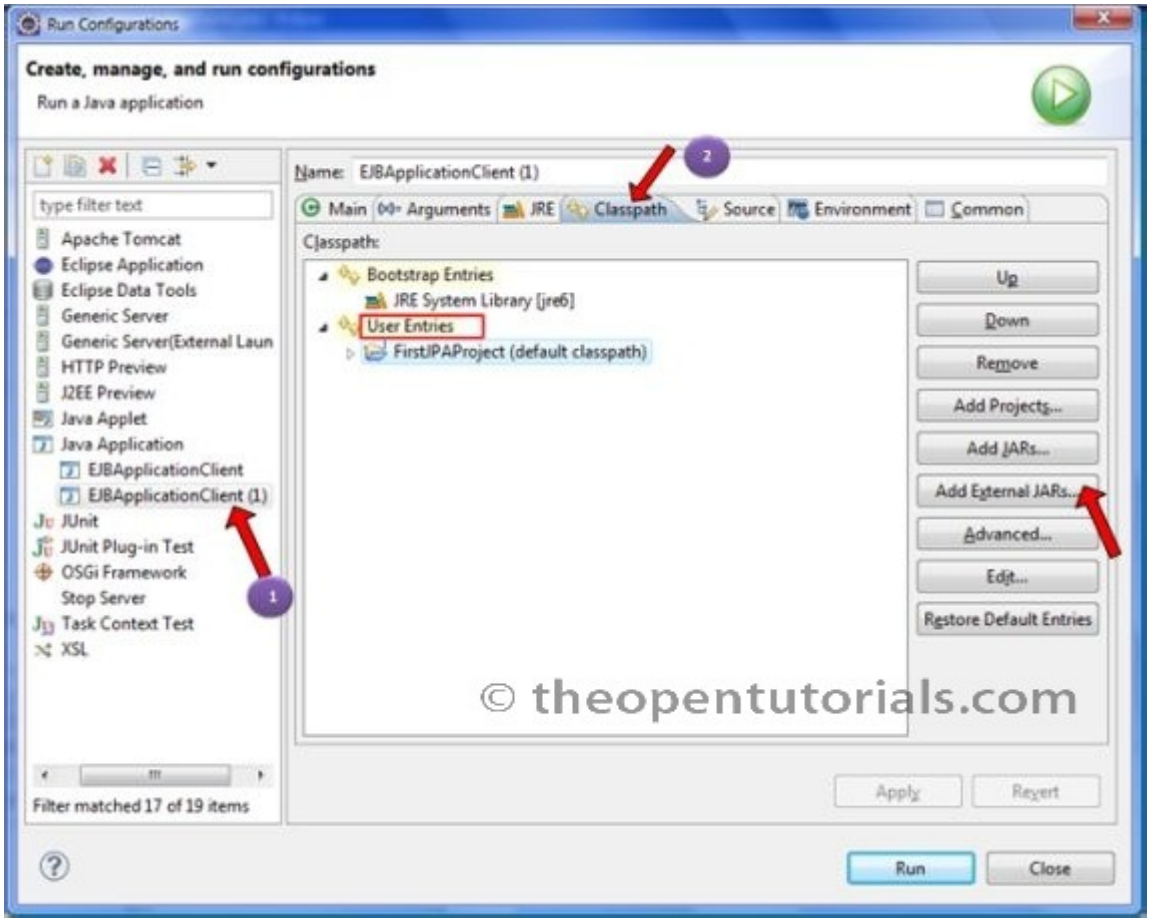
```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false

remote.connections=default

remote.connection.default.host=localhost
remote.connection.default.port = 4447
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
```

ADDING JAR FILES REQUIRED FOR THE CLIENT TO RUN THE CLIENT APPLICATION

- Open Run Conﬁgurations... in Run menu or Run Conﬁgurations in Run icon.



- Select the client application (EJBApplicationClient) under Java Application from left pane and open the Classpath tab from right side pane. If you don’t see your client application, run it once. Select “User Entries” and click on “Add External JARs”.

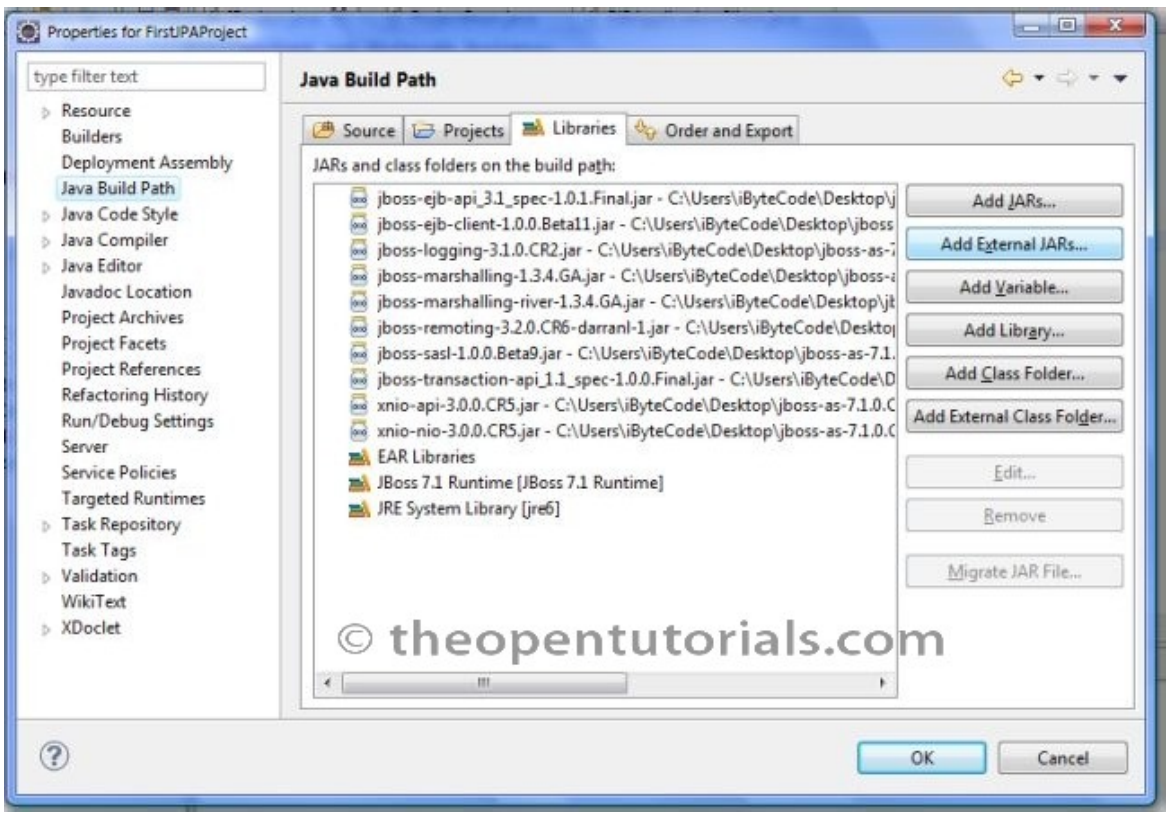


- Add the following JAR files.

JAR NAME	LOCATION
jboss-transaction-api_1.1_spec-1.0.0.Final.jar	AS7_HOME/modules/javax/transaction/api/main/
jboss-ejb-api_3.1_spec-1.0.1.Final.jar	AS7_HOME/modules/javax/ejb/api/main/
jboss-ejb-client-1.0.0.Beta10.jar	AS7_HOME/modules/org/jboss/ejb-client/main/
jboss-marshalling-1.3.0.GA.jar	AS7_HOME/modules/org/jboss/marshalling/main/
xnio-api-3.0.0.CR5.jar	AS7_HOME/modules/org/jboss/xnio/main/

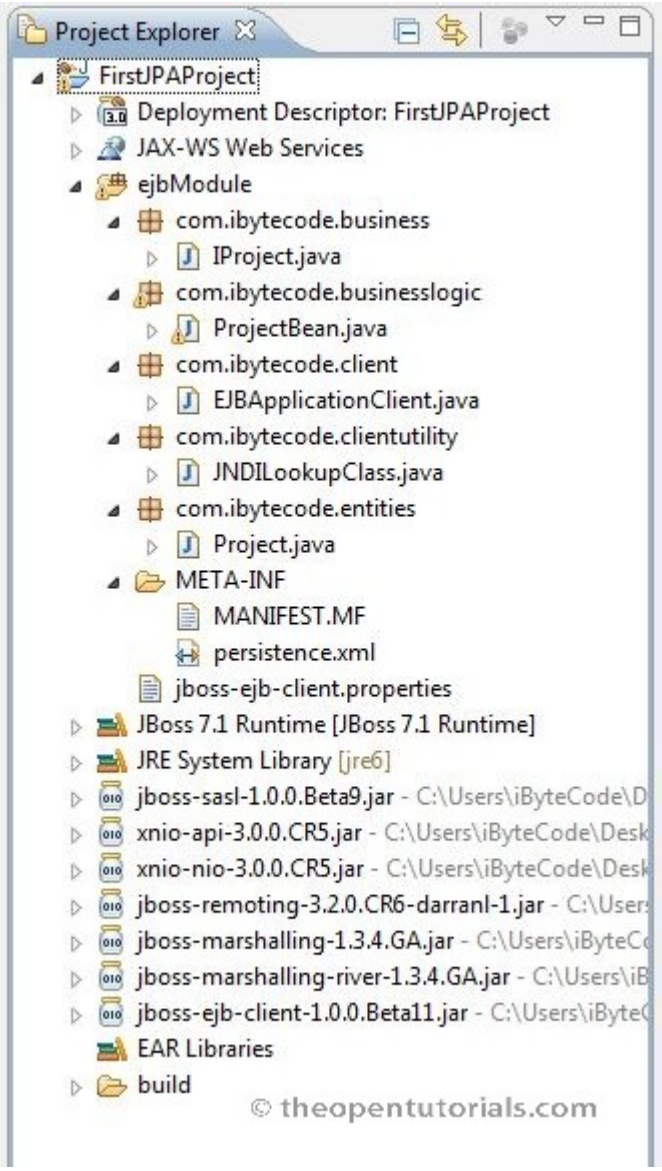
jboss-remoting-3.2.0.CR6.jar	AS7_HOME/modules/org/jboss/remoting3/main/
jboss-logging-3.1.0.Beta3.jar	AS7_HOME/modules/org/jboss/logging/main/
xnio-nio-3.0.0.CR5.jar	AS7_HOME/modules/org/jboss/xnio/nio/main/
jboss-sasl-1.0.0.Beta9.jar	AS7_HOME/modules/org/jboss/sasl/main/
jboss-marshalling-river-1.3.0.GA.jar	AS7_HOME/modules/org/jboss/marshalling/river/main/

You can also add it in Build path (Right click on your EJB Project->Properties, select Java Build Path from left side pane and select Libraries from right side and click on Add External JARs)



If you are using JBoss Application Server (AS) 7.1.0 Final version then it is suf﬿cient to add only one client JAR ﬿le (jboss-client-7.1.0.Final.jar) which is located in AS7_HOME/bin/client

The ﬿gure below shows the ﬿nal directory structure of this example.



RUN THE CLIENT

Use Ctrl + F11 to run the client.

Project [pnumber=1, pname=Banking App, plocation=Town City, deptNo=1] List of Projects:
Project [pnumber=1, pname=Banking App, plocation=Town City, deptNo=1] Project [pnumber=2, pname=Of﷽ce Automation, plocation=Downtown, deptNo=2]

Reference:

1. Praveen Macherla
2. https://ibytecode.com/blog/how-to-create-ejb3-jpa-project-in-eclipse-jboss-as-7-1/