TP 3 des architectures logicielles
Séance 3 : Architecture n-tiers distribuée à base d'EJB

1 Préparation de l'environnement Eclipse

1. Environment Used

- JDK 7 (Java SE 7)
- EJB 3.0
- Eclipse
- JBoss Tools – Core 4.4.1 M5 for Eclipse
- JBoss Application Server (AS) 7.1.0 Final

2. Installing JDK

JDK should be installed with proper environment set up. Read this page for installing the JDK and setting up the environment.

3. Installing Eclipse IDE

We use Eclipse IDE through out this tutorial. If you need to install Eclipse, you can read this page.

4. Installing JBoss Tools

JBoss Tools has set of Eclipse plug-ins that supports JBoss and related technology like Hibernate, JBoss AS, EJB and more… You can read this page to install JBoss Tools for Eclipse IDE.

5. Downloading JBoss Application Server (AS)

If you need to install JBoss AS, you can download it from this location: http://www.jboss.org/jbossas/downloads/

6. Installing JBoss Application Server (AS)

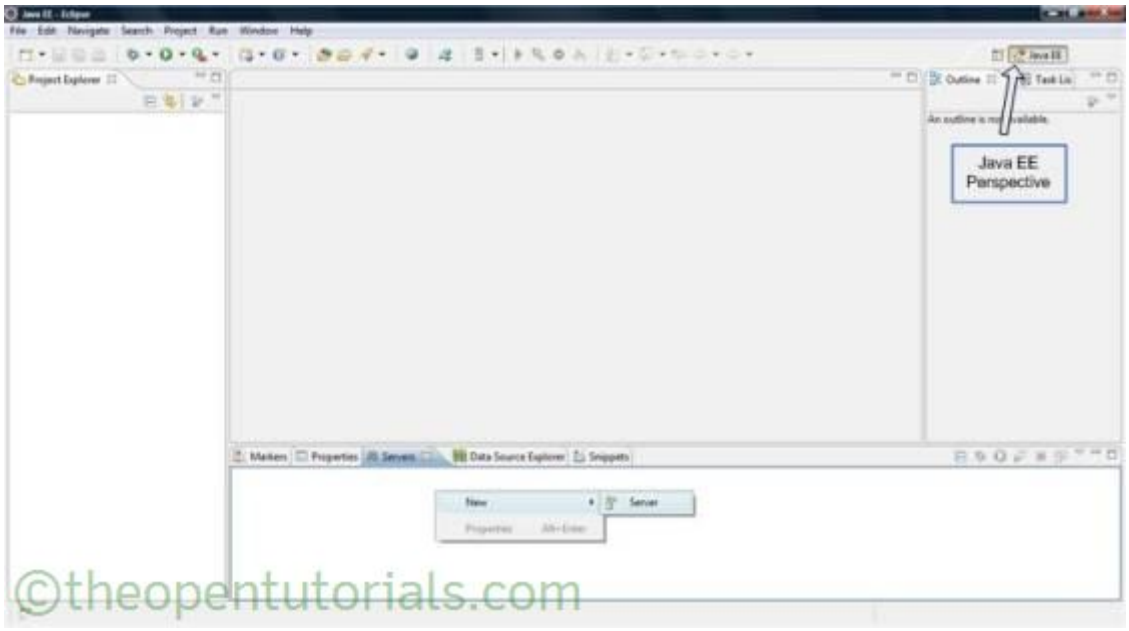
To install JBoss AS, all you have to do is simply extract the downloaded (zip or tar formats) file to a safe location on your machine. You can install JBoss Application Server on any operating system that supports the zip or tar formats.

7. Configuring JBoss AS in Eclipse IDE

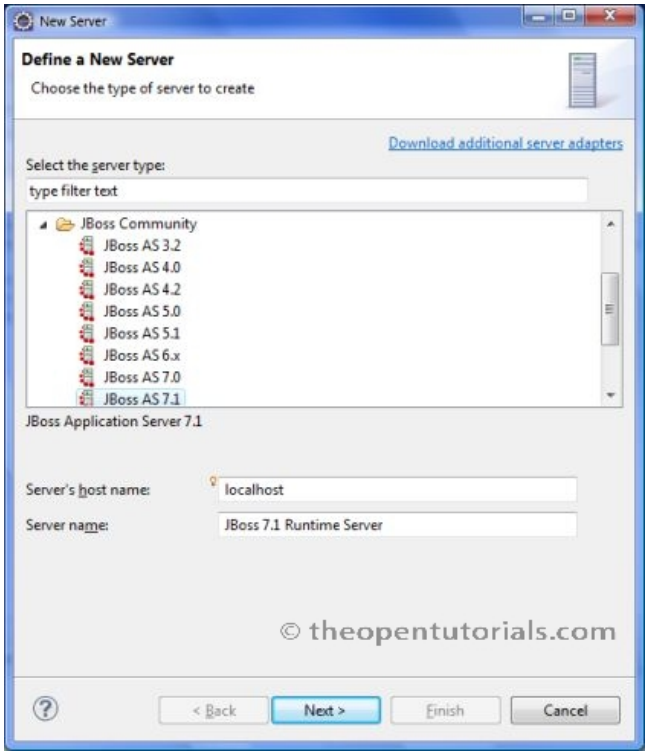
Open Eclipse IDE

Add Server

Make sure you are in Java EE perspective and in “Servers” area, right click -> New -> Server.



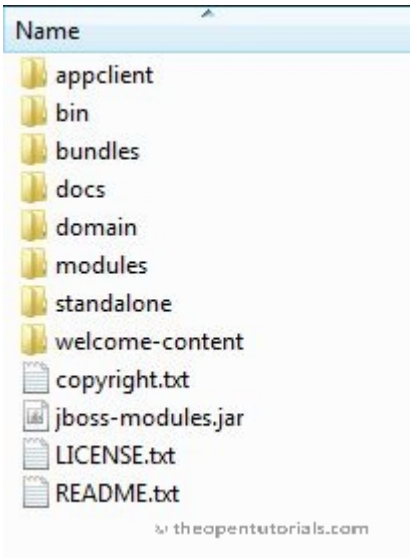
Here you will see list of servers that can be configured in the installed Eclipse version. You will find JBoss AS 7.1 under “JBoss Community” as shown below.



Select “JBoss AS 7.1” and click **Next**.

Configuring JBoss AS location

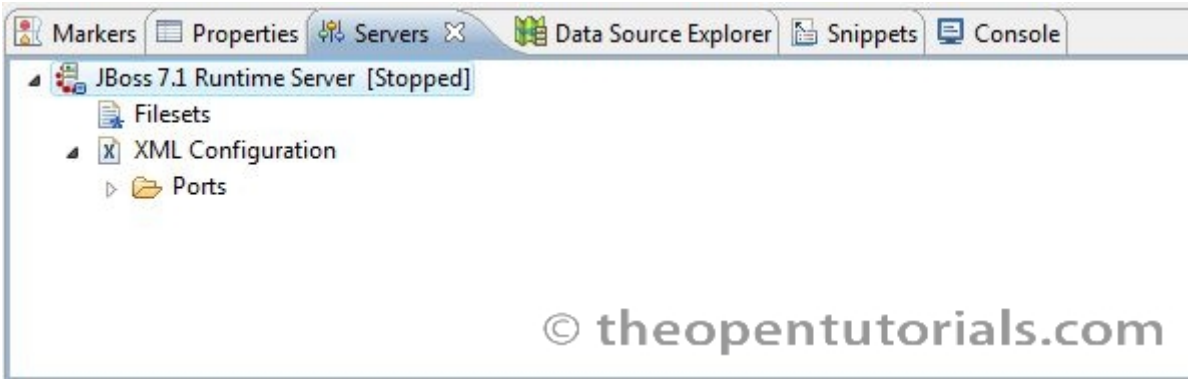
Select the JBoss Root folder which has bin folder.



and click **Next**.

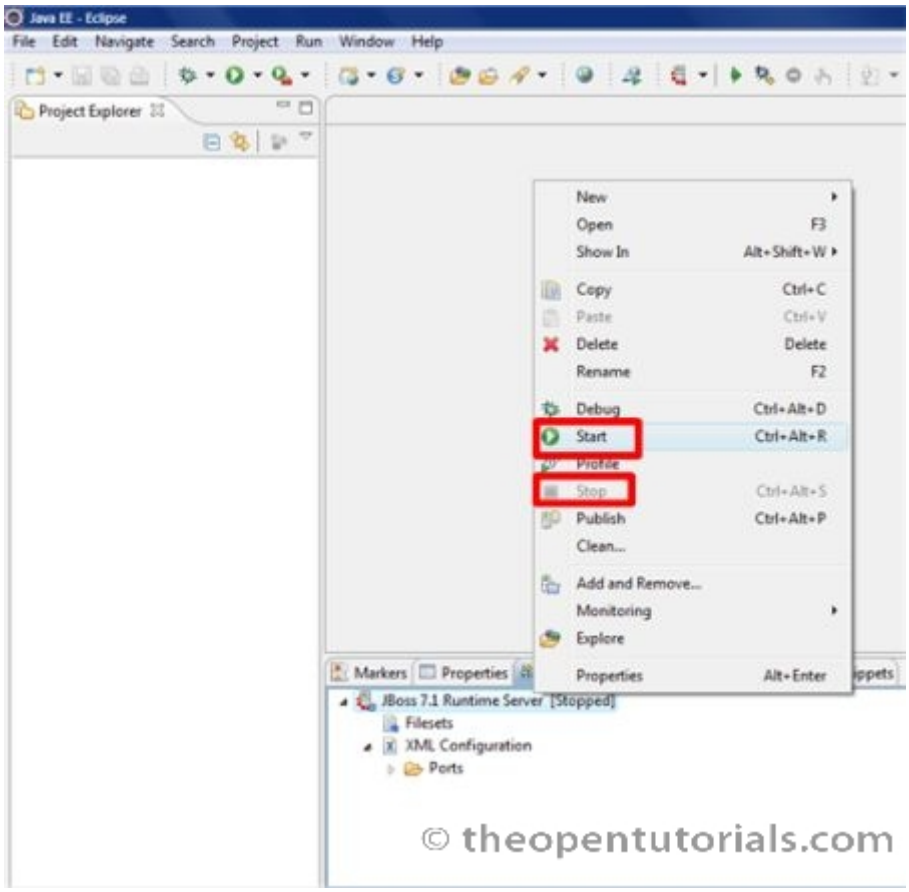
8. Finish

Make sure the runtime information is correct and click **Finish**. The configured JBoss AS will be displayed in the "Servers" view.



9. Start Server

It is easy to manage the server instance. Right-click on the server and start and stop it to ensure its proper working.

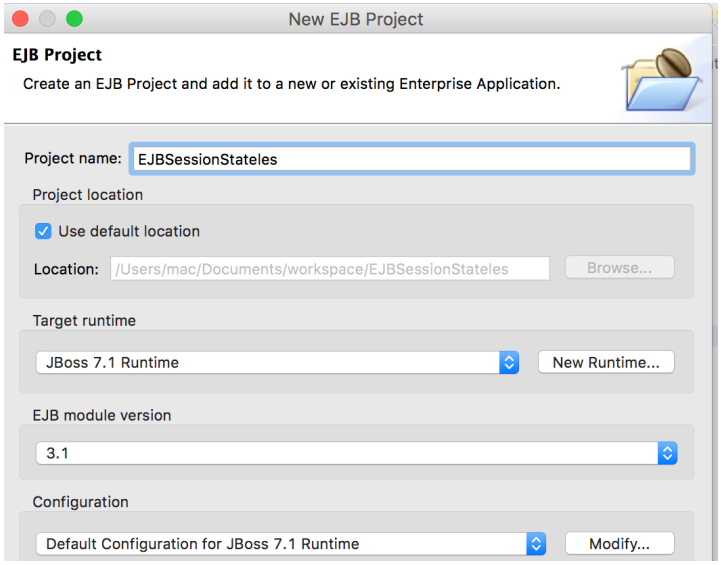


10. Test your Installation

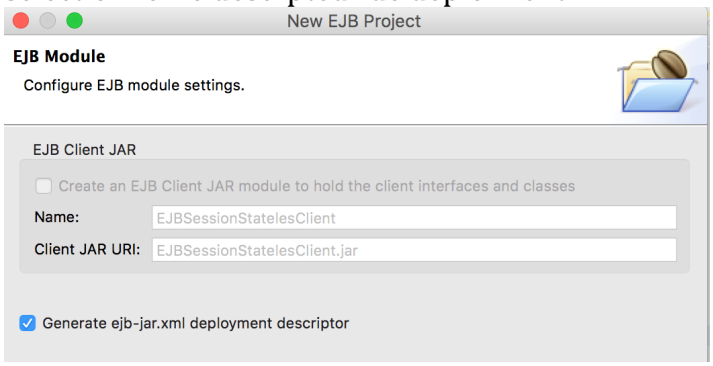
Start the server and type http://localhost:8080 in your browser and you should able to see the JBoss AS Welcome screen.

2 EJB session Stateless

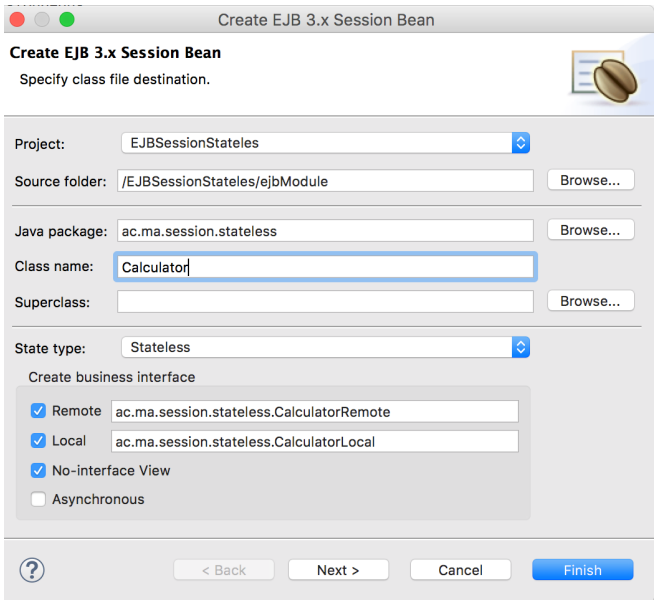
1. Créer un nouveau projet EJB 3.1 : EJBSessionStateless



2. selectionner le descrpteur de deploiment



3. Ajouter un session bean de type stateless



4. Implémenter les trois méthodes add(), mult() et Say() du bean Calculator

```
@Stateless
@LocalBean
public class Calculator implements CalculatorRemote, CalculatorLocal {

    /**
     * Default constructor.
     */
    public Calculator() {
        // TODO Auto-generated constructor stub
    }


        public float add(float a, float b) {
                // TODO Auto-generated method stub
                return a+b;
        }

        public float mult(float a, float b) {
                // TODO Auto-generated method stub
                return a*b;
        }

        public String Say(String s) {
                // TODO Auto-generated method stub
                return s;
        }
```

5. Déployer le projet sur le serveur JBoss AS
6. Préparation du client :
    a. Créer une nouvelle application Java : SimpleClientEJB
    b. Ajouter le projet EJBSessionStateless.jar au projet
    c. Ajouter la bibliothèque jboss-cleint.jar au projet
    d. Ajouter une nouvelle classe Java : CientCalculatorEJB
    e. Ajouter une méthode pour le paramétrage et l'initialisation du contexte d'accès au conteneur EJB :

```
public static Context getInitialContext() throws NamingException {
            Properties properties = new Properties();
            properties.put("jboss.naming.client.ejb.context", true);
            properties.put(Context.URL_PKG_PREFIXES, "org.jboss.ejb.client.naming");
            properties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jboss.naming.remote.client.InitialContextFactory");
            properties.put(Context.PROVIDER_URL,"remote://127.0.0.1:4447");
    properties.put("jboss.naming.client.connect.options.org.xnio.Options.SASL_POLIC
Y_NOPLAINTEXT", "false");
            //properties.put(Context.SECURITY_PRINCIPAL, "root");
            //properties.put(Context.SECURITY_CREDENTIALS, "password");
            return new InitialContext(properties);

    }
```

f. Ajouter une méthode pour la recherche du bean Calculator

```
private static CalculatorRemote lookupRemoteStatelessCalculator() throws
NamingException {
            Context context1 = ClientCalculatorEJB.getInitialContext();
            // The app name is the application name of the deployed EJBs. This is
typically the ear name
            // without the .ear suffix. However, the application name could be
overridden in the application.xml of the
            // EJB deployment on the server.
            // Since we haven't deployed the application as a .ear, the app name for
us will be an empty string
            final String appName = "";
            // This is the module name of the deployed EJBs on the server. This is
typically the jar name of the
            // EJB deployment, without the .jar suffix, but can be overridden via
the ejb-jar.xml
            // In this example, we have deployed the EJBs in a jboss-as-ejb-remote-
app.jar, so the module name is
            // jboss-as-ejb-remote-app
            final String moduleName = "EJBSessionStateless";
            // AS7 allows each deployment to have an (optional) distinct name. We
haven't specified a distinct name for
            // our EJB deployment, so this is an empty string
            final String distinctName = "";
            // The EJB name which by default is the simple class name of the bean
implementation class
            final String beanName = Calculator.class.getSimpleName();
            // the remote view fully qualified class name
            final String viewClassName = CalculatorRemote.class.getName();
            // let's do the lookup
            return (CalculatorRemote) context1.lookup("ejb:" + appName + "/" +
moduleName + "/" + distinctName + "/" + beanName + "!" + viewClassName);
    }
```

g. Ajouter une méthode pour l'invocation JNDI de l'EJB

```
private static void invokeStatelessBean() throws NamingException {
        // Let's lookup the remote stateless calculator
        final CalculatorRemote statelessRemoteCalculator =
lookupRemoteStatelessCalculator();
        System.out.println("Obtained a remote stateless calculator for
invocation");
        // invoke on the remote calculator
        int a = 100;
        int b = 200;
        System.out.println("Adding " + a + " and " + b + " via the remote stateless
calculator deployed on the server");
        float sum = statelessRemoteCalculator.add(a, b);
        System.out.println("Remote calculator returned sum = " + sum);
```

```
        // try one more invocation, this time for subtraction
        int num1 = 5;
        int num2 = 4;
        System.out.println("Multiplication " + num2 + " time " + num1 + " via the
remote stateless calculator deployed on the server");
        float mult = statelessRemoteCalculator.mult(num1, num2);
        System.out.println("Remote calculator returned multiplication = " + mult);

              }
```

h. Implanter la méthode principale

```
public static void main(String[] args) throws NamingException {
        System.out.println("Entering into the main method!!!");
        // Invoke a stateless bean
        invokeStatelessBean();
}
```

i. Exécuter votre client et vérifier le résultat.

3 EJB session stateful

1. Créer un nouveau projet EJB 3.1 : EJBSessionStateful
2. Ajouter un nouveau EJB session de type Stateful : Counter
3. Implémenter les méthodes : increment() et decrement() suivantes :

```
@Stateful
@LocalBean
public class Counter implements CounterRemote, CounterLocal {

public Counter() {
        // TODO Auto-generated constructor stub
    }

    private int count = 0;

    @Override
    public void increment() {
        this.count++;
    }

    @Override
    public void decrement() {
        this.count--;
    }

    @Override
    public int getCount() {
        return this.count;
    }

}
```

4. Suivre les mêmes étapes que le session bean stateless pour invoquer ce bean par un client Java.

```
private static void invokeStatefulBean() throws NamingException {
        // Let's lookup the remote stateful counter
        final CounterRemote statefulRemoteCounter = lookupRemoteStatefulCounter();
        System.out.println("Obtained a remote stateful counter for invocation");
        // invoke on the remote counter bean
        final int NUM_TIMES = 20;
        System.out.println("Counter will now be incremented " + NUM_TIMES + " times");
```

```
        for (int i = 0; i < NUM_TIMES; i++) {
            System.out.println("Incrementing counter");
            statefulRemoteCounter.increment();
            System.out.println("Count after increment is " +
statefulRemoteCounter.getCount());
        }
        // now decrementing
        System.out.println("Counter will now be decremented " + NUM_TIMES + " times");
        for (int i = NUM_TIMES; i > 0; i--) {
            System.out.println("Decrementing counter");
            statefulRemoteCounter.decrement();
            System.out.println("Count after decrement is " +
statefulRemoteCounter.getCount());
        }
    }
```

4 Extension de l'application n-tiers du TP 2

1. Ajouter un EJB session, permettant de calculer le nombre d'heures enseigné par un professeur donné.
    a. Le nom du professeur est sélectionné à travers une page JSP.
    b. Le contrôle et l'accès à l'EJB doit se faire par une Servlet.
    c. L'affichage est effectué par un JSP.