JBossIDE Tutorial

An introduction and walkthrough of JBossIDE

1.4

Table of Contents

Preface

1. Foreword

JBoss-IDE started with an XDoclet plug-in for eclipse in the middle of 2002. Then Hans Dockter met Marc as he participated at a JBoss training in Mallorca and they talked about the possibility of developing a JBoss-IDE.

2. About the Authors

- Marshall Culpepper is the project lead of JBoss-IDE. Marshall is a full time employee of JBoss, Inc. And lives in Dallas, Texas.
- Laurent Etiemble, is an active contributor to the JBoss-IDE project. Laurent works as a consultant and lives in Paris, France.
- Hans Dockter, was the founder and lead architect of the Jboss-IDE project. Hans works as an independent consultant and lives in Berlin, Germany.

3. About JBoss

JBoss Project, headed by Marc Fleury, is composed of over 100 developers worldwide who are working to deliver a full range of J2EE tools, making JBoss the premier Enterprise Java application server for the Java 2 Enterprise Edition platform.

JBoss is an Open Source, standards-compliant, J2EE application server implemented in 100% Pure Java. The JBoss/Server and complement of products are delivered under a public license. With a huge amount of downloads per month, JBoss is the most downloaded J2EE based server in the industry.

4. Acknowledgements

We would like to thank Thomas Deichsel and Frank Henze from media-style.com for their wonderful interface design. We would also like to thank all the JBoss-IDE community for their support and their feedback.

1

Introduction to JBossIDE

JBossIDE offers you:

- Extensive and intuitive support for XDoclet.
- The debugging and monitoring of JBoss servers and the controlling of their life cycles.
- An easy way to configure the packaging layout of archives (packed or exploded)
- A simple way to deploy the packaged and/or exploded archive to a JBoss server
- Several J2EE wizards to ease and simplify J2EE development.
- Source code editors for JSP, HTML, and XML

2

Tutorial Preparation

2.1. Introduction

The goal of this tutorial is to demonstrate how simple it is to develop J2EE applications with JBossIDE. The sample application that will be built is a J2EE application with one session EJB and one Servlet, which computes the Fibonacci suite.

The tutorial is split into several parts:

- The Project: this part shows how the project is prepared (source and build path)
- The EJB: this part shows how to write an EJB class with its XDoclet javadoc tags.
- Generation of EJB files: this part shows how to configure the XDoclet generation configuration to generate all the EJB related files
- The Servlet and the Web-App: this part shows how to write a Servlet class with its XDoclet javadoc tags.
- Generation of Servlet files: this part shows how to configure the XDoclet generation configuration to generate all the Web related files
- The J2EE application: this part shows how to create the missing files.
- Packaging: this part shows how to package the J2EE application
- JBoss configuration : this part shows how to define debug configuration to launch JBoss inside Eclipse.
- Deployment : this part shows how to deploy by copy the J2EE application
- Debugging: this part shows how to set up breakpoints to debug the deployed application.

2.2. Requirements

For this tutorial you need:

- Java Development Kit 1.3.0 or higher (a JDK is needed to launch JBoss 3.x)
- Eclipse 3.0 (from eclipse.org [http://www.eclipse.org]) or higher.
- JBoss Application Server 3.x-4.x

You also will need to know about developing and debugging applications in Eclipse. Refer to the Eclipse website [http://www.eclipse.org] for further informations.

2.3. Notes

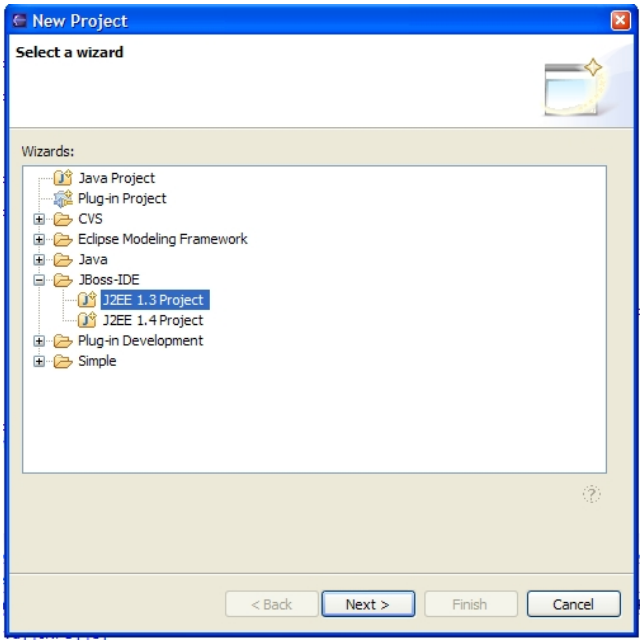
This tutorial has been written using the Eclipse 3.0 series. If you are using the Eclipse 3.1 series, the screens may be slightly different.

3

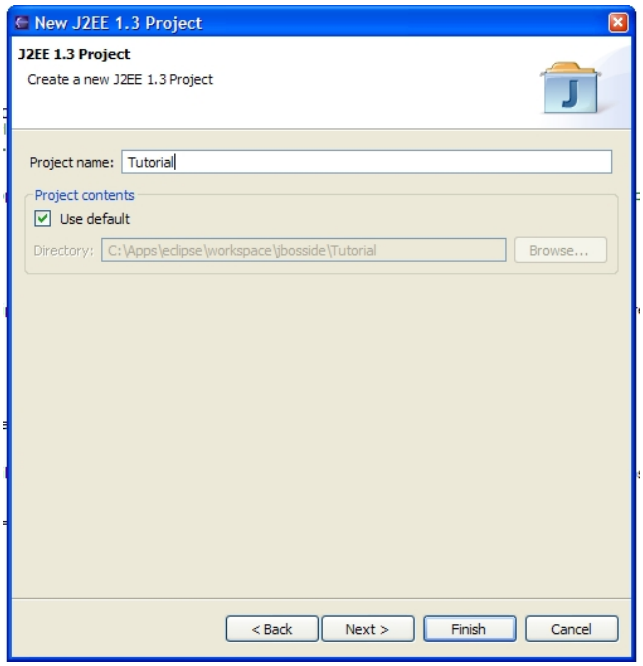
The Project

We will create a source folder, import libraries and make the build path.

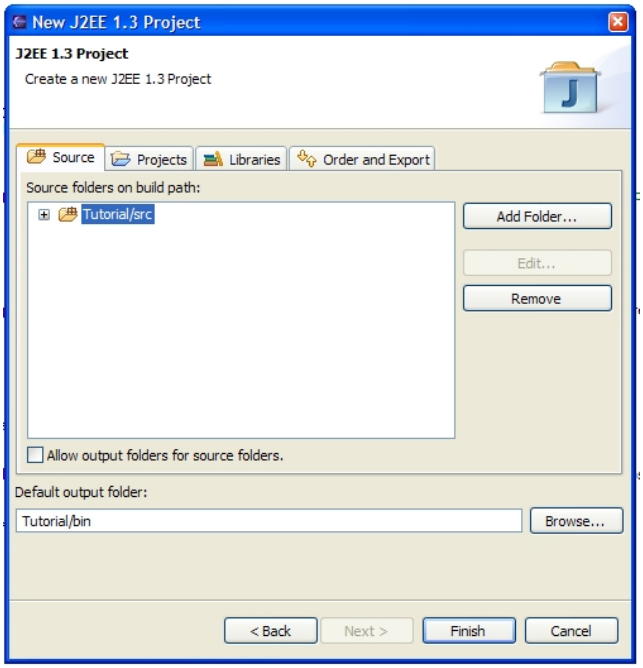
Create a new J2EE 1.3 Project. Select File > `New` > `Project...` and choose `JBoss-IDE` > `J2EE 1.3 Project`.



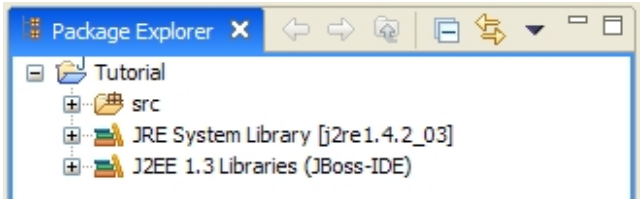
Enter `Tutorial` for the project name and select the Next button.



Create a source folder named `src`. Make sure the default output folder will be `bin`.



In the package explorer, the new project should look like this. Note that the J2EE 1.3 core classes are directly added. They are available like a standard library with all the source code.

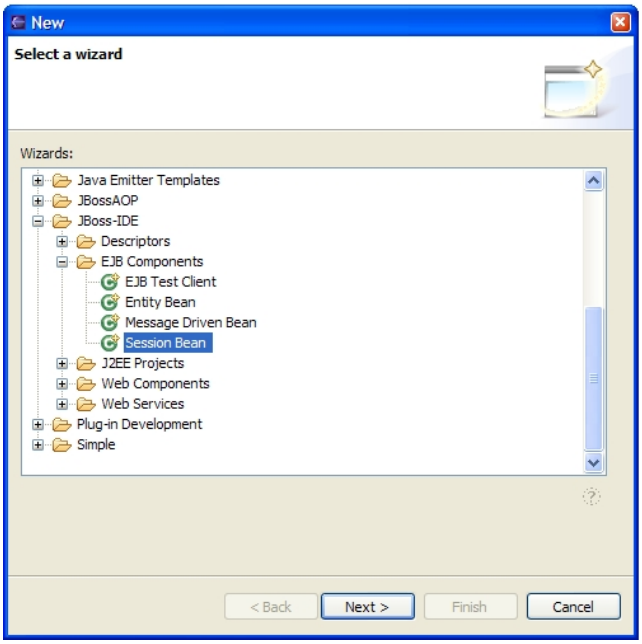


4

The EJB

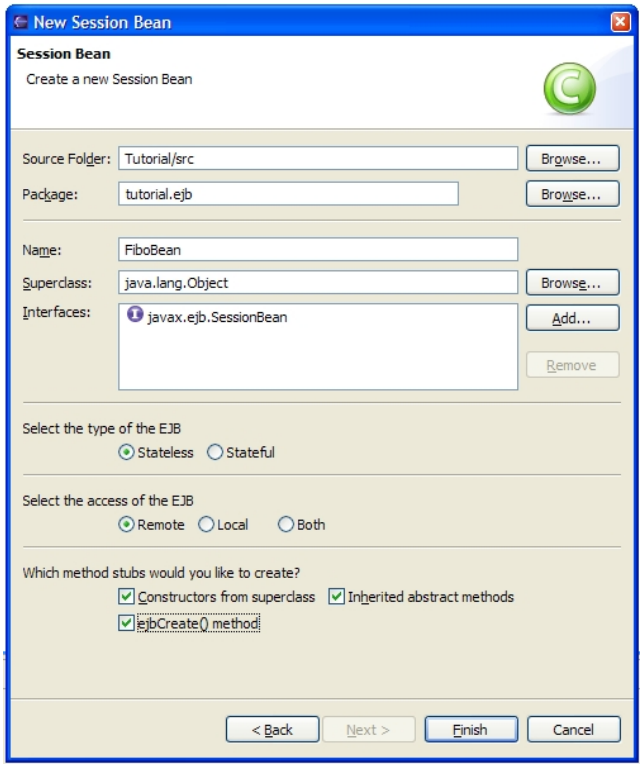
The next step is to create an EJB. For simplicity, it will be a stateless session bean, but others types are also easy to write.

Create a new Session EJB. Select `File` > `New` > `Other...` and choose `JBoss-IDE` > `EJB Components` > `Session Bean`.

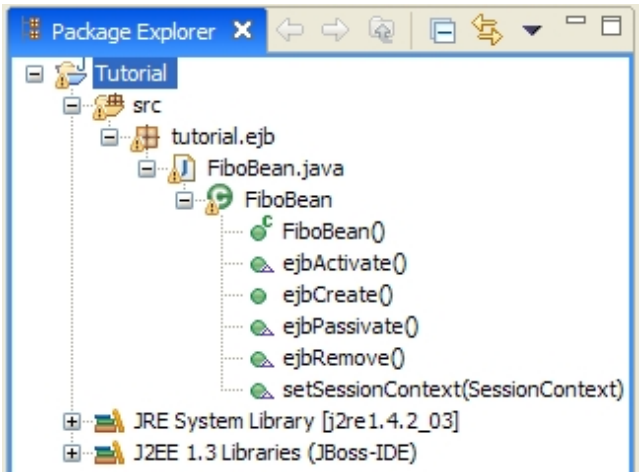


The package will be `tutorial.ejb` and the class name “FiboBean”.

Leave the default options selected and be sure that `ejbCreate() method` is checked.

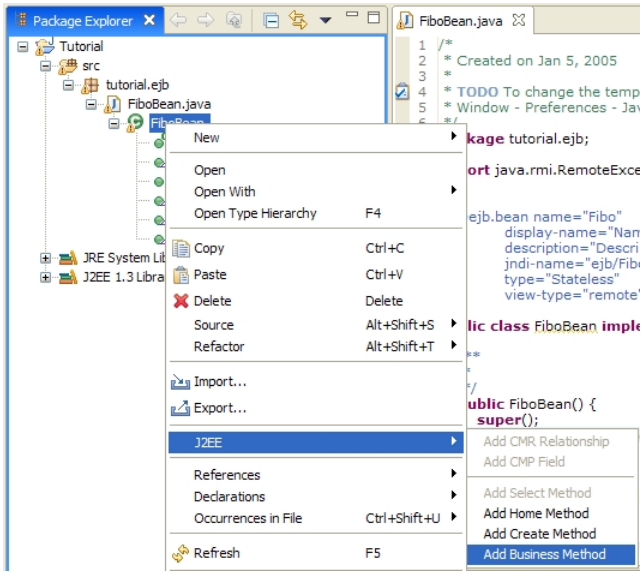


Click on “Finish”. The class is then created and you should have a project like this. Note that all the method stubs are created with the default ejbCreate method.

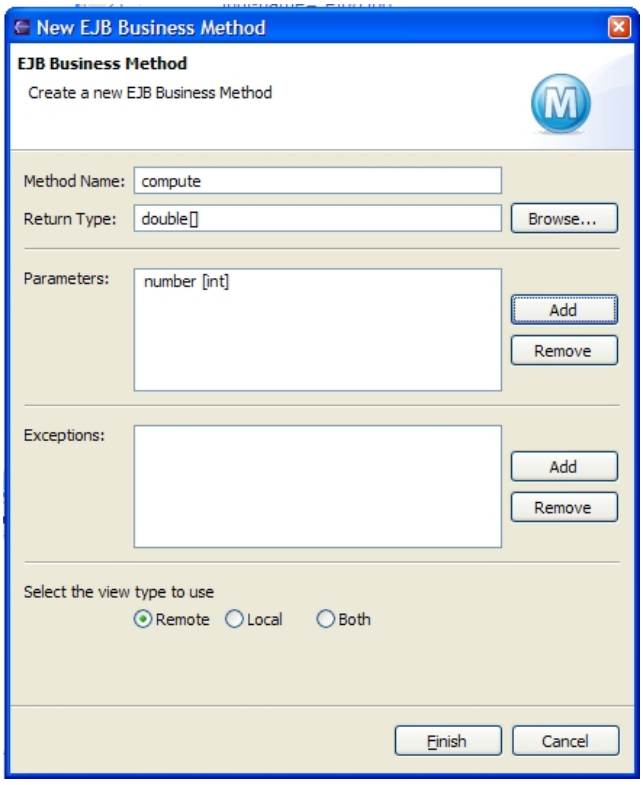


To make this interesting, we will create a business method for our EJB that computes a Fibonacci suite.

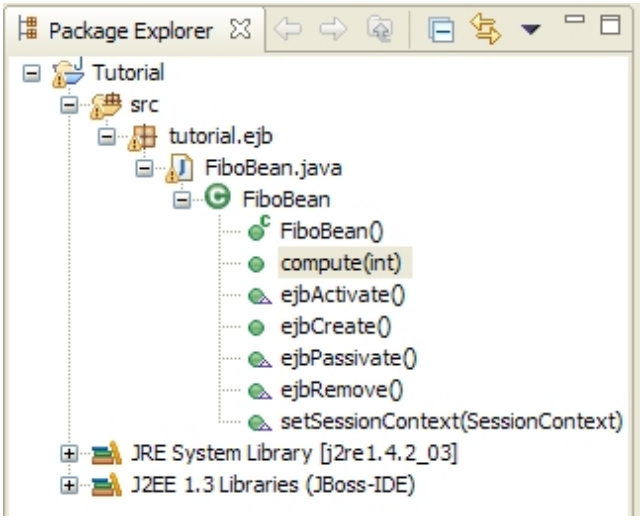
Right-click the FiboBean class, under the FiboBean Java file. You should see a `J2EE` menu. Select `J2EE` > `Add Business Method`.



In the method wizard, enter `compute` as the method name, `double[]` for the return type and add a parameter called `number` of type `int`. Click on Finish.



A new method has been added to the FiboBean class.



In the text editor, complete the body of the compute method as below :

```
public double[] compute(int number) {
    if (number < 0) {
        throw new EJBException("Argument should be positive");
    }

    double[] suite = new double[number + 1];
    suite[0] = 0;

    if (number == 0) {
        return suite;
    }

    suite[1] = 1;

    for (int i = 2; i <= number; i++) {
        suite[i] = suite[i - 1] + suite[i - 2];
    }

    return suite;
}
```

As you may have noticied, each wizard adds all of the required XDoclet tags. Go to the top of the class and complete the attributes of the tag with the following values (by pressing CTRL+Space for each attribute, you will get an auto-compled list) :

```
/**
 * @ejb.bean  name="Fibo"
 *            display-name="Name for Fibo"
 *            description="Description for Fibo"
 *            jndi-name="ejb/Fibo"
 *            type="Stateless"
 *            view-type="remote"
 */
public class FiboBean implements SessionBean {
```

After that, the file should look like this. Now, we are ready to run XDoclet on the file to generate the EJB interfaces.

```
package tutorial.ejb;

import java.rmi.RemoteException;

import javax.ejb.EJBException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

import javax.ejb.CreateException;

/**
 * @ejb.bean  name="Fibo"
 *            display-name="Name for Fibo"
 *            description="Description for Fibo"
 *            jndi-name="ejb/Fibo"
 *            type="Stateless"
 *            view-type="remote"
 */
public class FiboBean implements SessionBean {

    /**
     *
     */
    public FiboBean() {
        super();
        // TODO Auto-generated constructor stub
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.ejb.SessionBean#ejbActivate()
     */
    public void ejbActivate() throws EJBException, RemoteException {
        // TODO Auto-generated method stub
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.ejb.SessionBean#ejbPassivate()
     */
    public void ejbPassivate() throws EJBException, RemoteException {
        // TODO Auto-generated method stub
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.ejb.SessionBean#ejbRemove()
     */
    public void ejbRemove() throws EJBException, RemoteException {
        // TODO Auto-generated method stub
    }

    /*
     * (non-Javadoc)
     *
     * @see javax.ejb.SessionBean#setSessionContext(javax.ejb.SessionContext)
     */
    public void setSessionContext(SessionContext ctx) throws EJBException,
```

```
            RemoteException {
        // TODO Auto-generated method stub
    }

    /**
     * Default create method
     *
     * @throws CreateException
     * @ejb.create-method
     */
    public void ejbCreate() throws CreateException {
        // TODO Auto-generated method stub
    }

    /**
     * Business method
     *
     * @ejb.interface-method view-type = "remote"
     */
    public double[] compute(int number) {
        if (number < 0) {
            throw new EJBException("Argument should be positive");
        }

        double[] suite = new double[number + 1];
        suite[0] = 0;

        if (number == 0) {
            return suite;
        }

        suite[1] = 1;

        for (int i = 2; i <= number; i++) {
            suite[i] = suite[i - 1] + suite[i - 2];
        }

        return suite;
    }
}
```

5

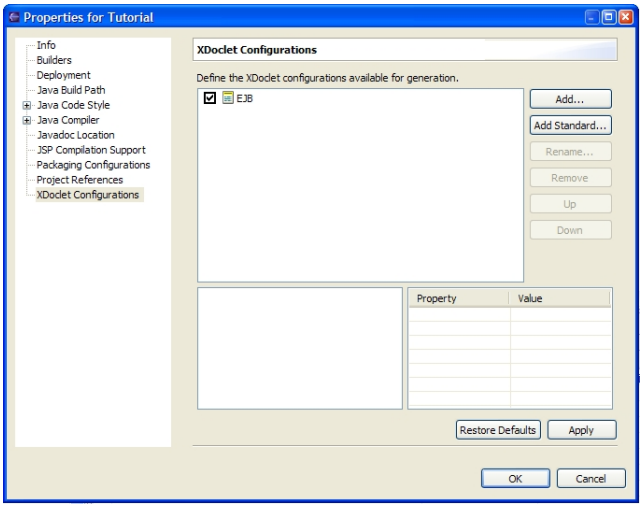
Generation of the EJB related files

To generate the EJB related classes and descriptors, we need to create some XDoclet configurations. With JBoss IDE, you can define several XDoclet generation configurations that will be run against the project.

Procedure 5.1. XDoclet EJB Configuration Creation

1. Edit the project properties by right clicking on the project and select `Properties`.
2. In the property page, select `XDoclet configurations`.
3. Right-click in the upper area to pop-up the menu and choose `Add`. Type `EJB` in the dialog and click `OK`.

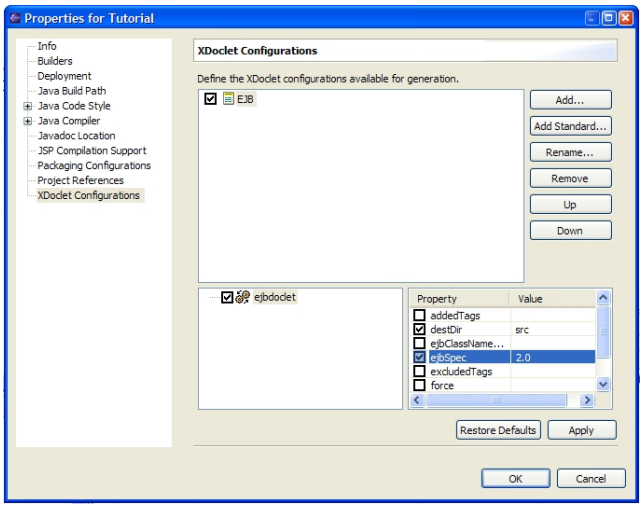
You have created a new generation configuration named `EJB`.



Procedure 5.2. Ejbdoclet Configuration

1. Select the `EJB` configuration.
2. In the lower-left area, right-click to popup the menu and choose `Add Doclet`.
3. A list of available doclets will appear. Choose `ejbdoclet` and click `OK`.
4. On the lower-right area, you see the properties of the `ejbdoclet`.
   a. Set the `destDir` property to `src`.
   b. Set the `ejbSpec` property to `2.0`.

Our configuration now contains an `ejbdoclet` that

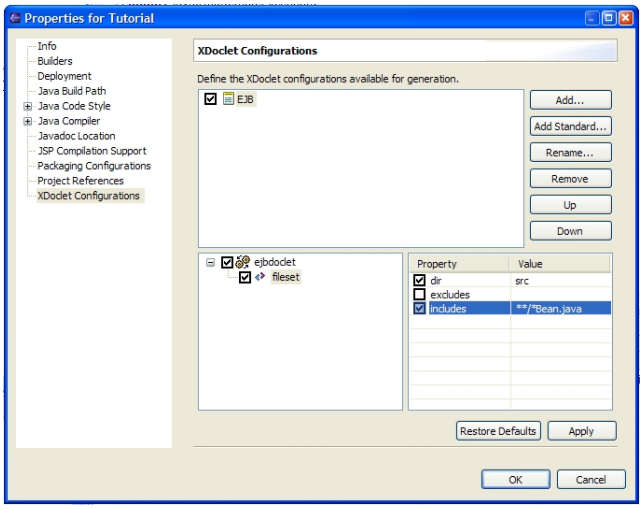


will produce files in `src` folder and for the EJB 2.0 specifications.

Procedure 5.3. Fileset Configuration

1. In the lower-left area, right-click on `ejbdoclet` to popup the menu and choose `Add`.
2. A list of available subtasks will appear. Choose `fileset` and click `Ok`.
3. On the lower-right area, you see the properties of the fileset.
    a. Set the `dir` property to `src`.
    b. Uncheck `excludes`
    c. Set the `includes` property to `**/*Bean.java`.

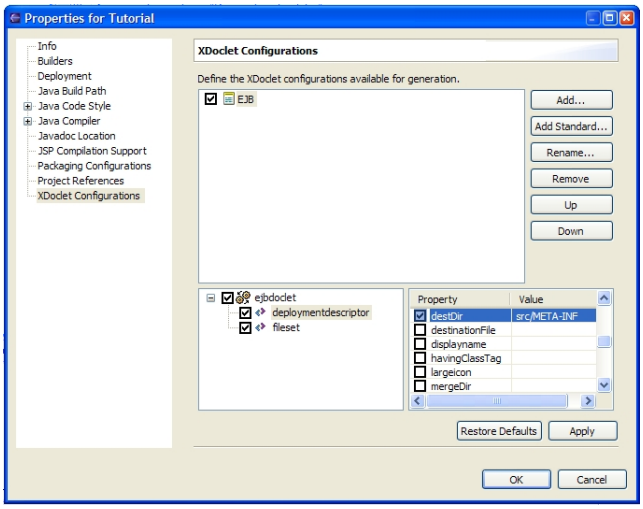
Our configuration now contains an `ejbdoclet` with a `fileset` that contains the `src` directory, and all files under it that end in `Bean.java`.



Procedure 5.4. Deployment Descriptor Configuration

- Add a new `deploymentdescriptor` subtask to the `ejbdoclet` (see above).
    - Set the `destDir` property to `src/META-INF`.

All of the standard EJB deployment descriptors will now be placed in the `src/META-INF` directory.



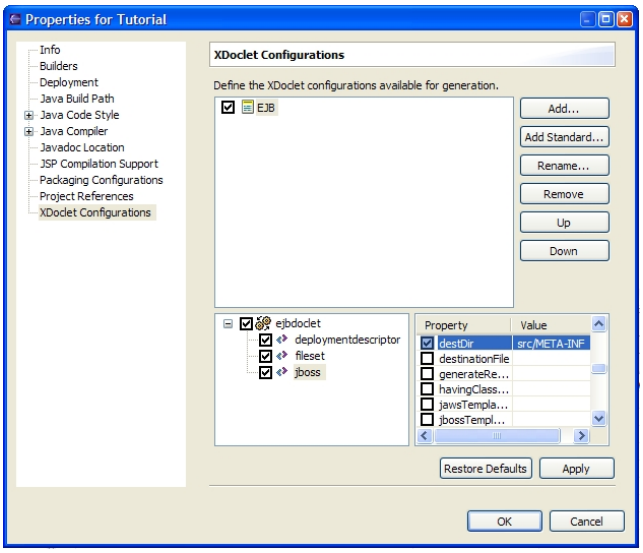
Procedure 5.5. JBoss Configuration

- Add a new `jboss` subtask to the `ejbdoclet` (see above).

  a. Set the `destDir` property to `src/META-INF`.

  b. Set the `Version` property to `3.0`.

All of the JBoss-specific deployment descriptors will now be placed in the `src/META-INF` directory.



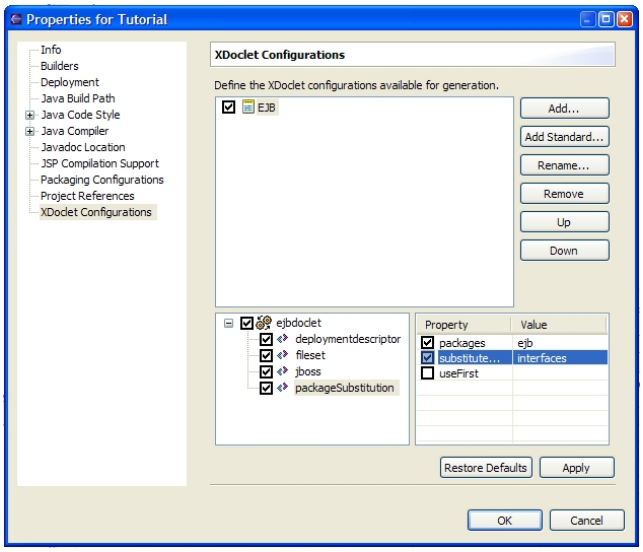
Procedure 5.6. Package Substitution Configuration

- Add a new `packageSubstitution` subtask to the `ejbdoclet` (see above).

  a. Set the `packages` property to `ejb`.

  b. Set the `substituteWith` property to `interfaces`.

This will place our generated EJB interfaces in the `tutorial.interfaces` java package.

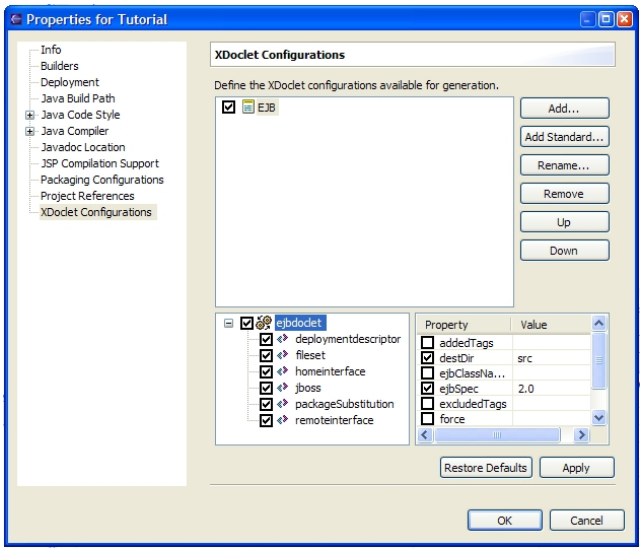


Procedure 5.7. Interface Configuration

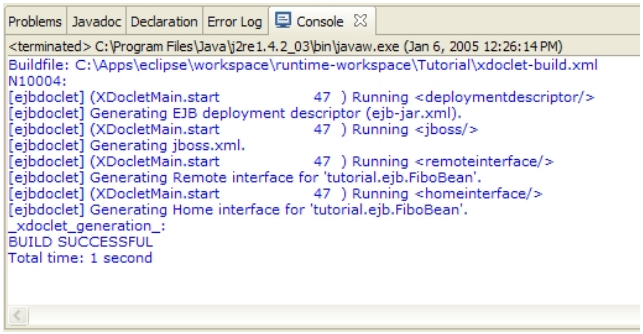
1. Add a new `remoteInterface` subtask to the `ejbdoclet` (see above).

2. Add a new `homeInterface` subtask to the `ejbdoclet` (see above).

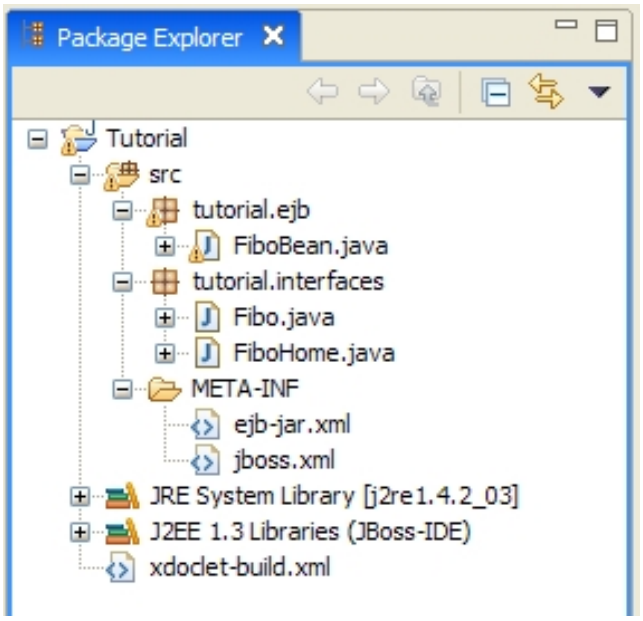
These subtasks will generate the EJB home and remote interfaces.



Click `OK` and the XDoclet configuration for the `Tutorial` will be saved. Once the configuration is saved, right-click on the `Tutorial` project and select `Run XDoclet`. The XDoclet generation will display its output in the console. The output should look like this:



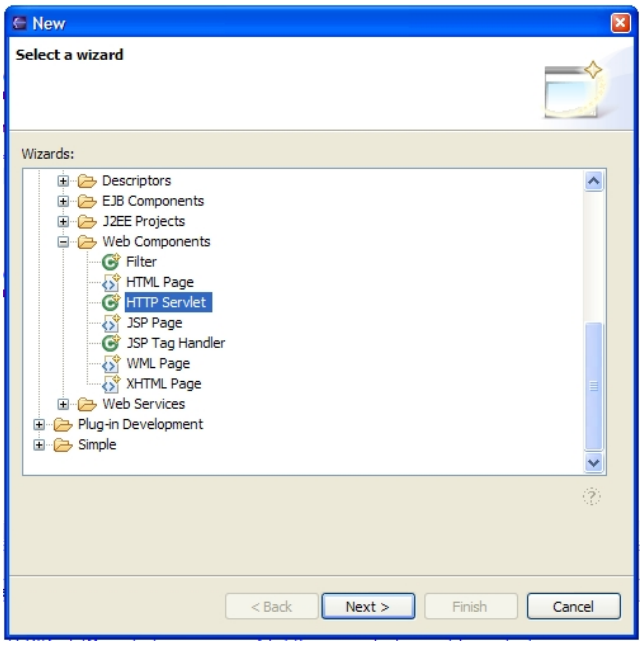
After the code generation, select the project and refresh it (you can press F5). You should have a project that looks like this. Note that a `tutorial.interfaces` package has been created with new classes inside. There is also a `META-INF` folder with the deployment descriptors (both standard and jboss).



6

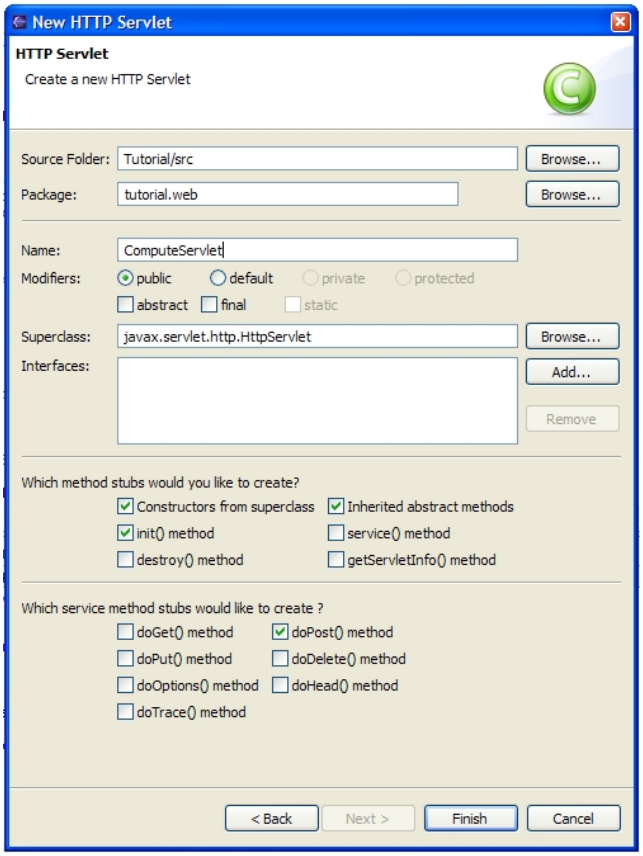
The Servlet and the Web-App

Having an EJB is not enough. We will write a servlet that access this EJB to perform the actual computation of the Fibonacci suite.

Create a new HTTP Servlet. Select `File` > `New` > Other... and choose `JBoss-IDE` > `Web Components` > `HTTP Servlet`.	

Procedure 6.1. HTTP Servlet Configuration

1. Set the `Package` to `tutorial.web`.
2. Set the Class `Name` to `ComputeServlet`.
3. Under `Which method stubs would you like to create?`, check the `init() method`.
4. Under `Which service method stubs would like to create?`, check the `doPost() method`.



Our servlet needs some initialization and processing code. Add the following private member.

```
                private FiboHome home;
```

Complete the `init` method as shown. This code is responsible for the initialization of the EJB Home interface and grabbing the local environment entry.

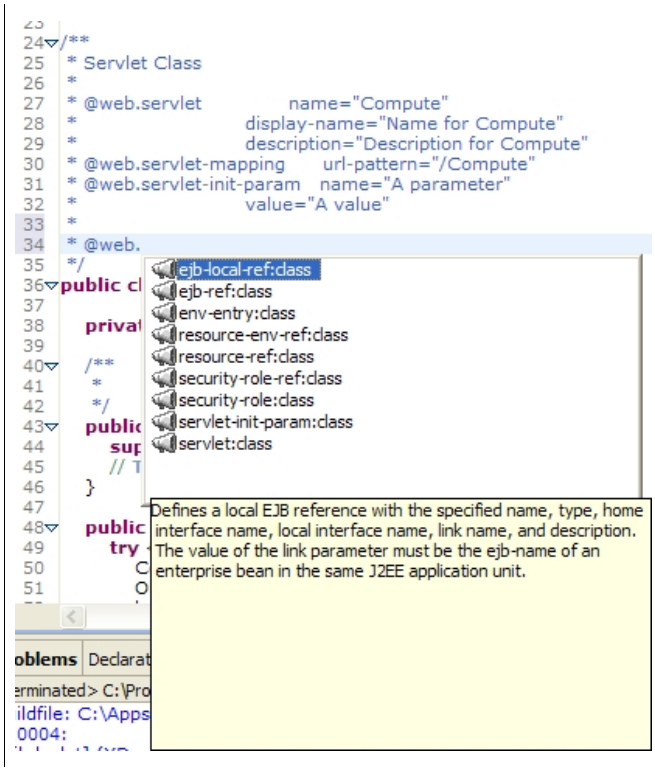
```
public void init(ServletConfig config) throws ServletException {
        try {
                Context context = new InitialContext();
                Object ref = context.lookup("java:/comp/env/ejb/Fibo");
                home = (FiboHome) PortableRemoteObject.narrow(ref, FiboHome.class);
        } catch (Exception e) {
                throw new ServletException("Lookup of java:/comp/env/ failed");
        }
}
```

Complete the `doPost` method as shown. The code will parse the request to get the limit parameter, create an instance of the EJB, perform computation, release the instance and output the result as HTML.

```
protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {

        response.setContentType("text/html");
```

```
        PrintWriter out = response.getWriter();

        out.println("<html><head><title>");
        out.println("Fibonaci Computation");
        out.println("</title></head>");
        out.println("<body>");

        out.println("<h1>");
        out.println("Fibonaci Computation");
        out.println("</h1>");

        try {
                Fibo bean = home.create();
                int limit = 0;
                String value = request.getParameter("limit");
                if (value != null) {
                        try {
                                limit = Integer.parseInt(value);
                        } catch (Exception e) {
                        }
                }
                double[] result = bean.compute(limit);
                bean.remove();

                out.println("<p>");
                out.print("The ");
                out.print(limit);
                out.print(" first Fibonacci numbers ");

                for (int i = 0; i < result.length; i++) {
                        out.println("<br>");
                        out.println(i);
                        out.println(" : ");
                        out.println(result[i]);
                }

                out.println("</p>");
        } catch (Exception e) {
                out.println(e.getMessage());
                e.printStackTrace(out);
        } finally {
                out.println("</body></html>");
                out.close();
        }
}
```

Next, we will insert the missing XDoclet tags for the Servlet. In the Java editor go in the Javadoc class paragraph. Type “@web.” And press CTRL+Space. You should see JBossIDE's auto-completion in action.



Correct and complete the attributes of the tag with the following values (press CTRL+Space for each attribute if you want the completion) :

```
/**
 * @web.servlet
 *          name="Compute"
 *          display-name="Computation Servlet"
 *          description="Servlet that compute Fibonacci suite"
 *
 * @web.servlet-mapping
 *                  url-pattern="/Compute"
 *
 * @web.ejb-ref
 *          name="ejb/Fibo"
 *          type="Session"
 *          home="tutorial.interfaces.FiboHome"
 *          remote="tutorial.interfaces.Fibo"
 *          description="Reference to the Fibo EJB"
 *
 * @jboss.ejb-ref-jndi
 *          ref-name="ejb/Fibo"
 *          jndi-name="ejb/Fibo"
 */
public class ComputeServlet extends HttpServlet {
```

After that, the file should look like this. Now we are ready to run XDoclet on the file, which will generate the Web descriptors.

```
package tutorial.web;

import java.io.IOException;
```

```
import java.io.PrintWriter;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import tutorial.interfaces.Fibo;
import tutorial.interfaces.FiboHome;

/**
 * @web.servlet
 *         name="Compute"
 *         display-name="Computation Servlet"
 *         description="Servlet that compute Fibonacci suite"
 *
 * @web.servlet-mapping
 *                 url-pattern="/Compute"
 *
 * @web.ejb-ref
 *         name="ejb/Fibo"
 *         type="Session"
 *         home="tutorial.interfaces.FiboHome"
 *         remote="tutorial.interfaces.Fibo"
 *         description="Reference to the Fibo EJB"
 *
 * @jboss.ejb-ref-jndi
 *         ref-name="ejb/Fibo"
 *         jndi-name="ejb/Fibo"
 */
public class ComputeServlet extends HttpServlet {
        private FiboHome home;

        public ComputeServlet() {
                super();
        }

      public void init(ServletConfig config) throws ServletException {
                try {
                        Context context = new InitialContext();
                        Object ref = context.lookup("java:/comp/env/ejb/Fibo");
                        home = (FiboHome) PortableRemoteObject.narrow(ref, FiboHome.class);
                } catch (Exception e) {
                        throw new ServletException("Lookup of java:/comp/env/ failed");
                }
        }

        protected void doPost(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();

                out.println("<html><head><title>");
                out.println("Fibonaci Computation");
                out.println("</title></head>");
                out.println("<body>");

                out.println("<h1>");
                out.println("Fibonaci Computation");
                out.println("</h1>");

                try {
```

```
                        Fibo bean = home.create();
                        int limit = 0;
                        String value = request.getParameter("limit");
                        if (value != null) {
                                try {
                                        limit = Integer.parseInt(value);
                                } catch (Exception e) {
                                }
                        }
                        double[] result = bean.compute(limit);
                        bean.remove();

                        out.println("<p>");
                        out.print("The ");
                        out.print(limit);
                        out.print(" first Fibonacci numbers ");

                        for (int i = 0; i < result.length; i++) {
                                out.println("<br>");
                                out.println(i);
                                out.println(" : ");
                                out.println(result[i]);
                        }

                        out.println("</p>");
                } catch (Exception e) {
                        out.println(e.getMessage());
                        e.printStackTrace(out);
                } finally {
                        out.println("</body></html>");
                        out.close();
                }
        }
}
```

7

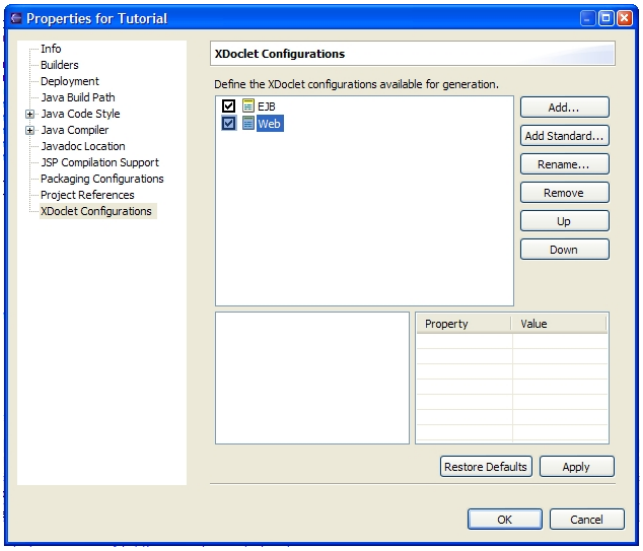
Generation of the Servlet related files

To generate the Web descriptors, we need to create another XDoclet configuration, like we did for our EJB.

Procedure 7.1. XDoclet Web Configuration Creation

1. Edit the project properties. Right-click on the project and select `Properties`.
2. In the properties page, select `XDoclet Configurations`.
3. Right-click in the top area to pop-up the menu and choose `Add`. Type `Web` in the dialog and click `OK`.

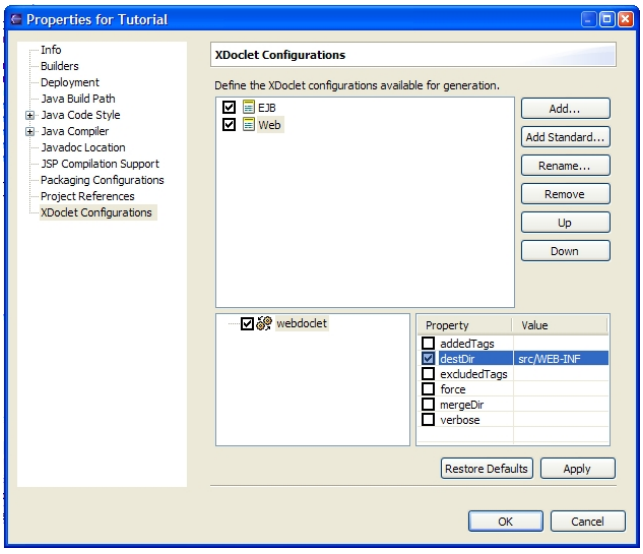
You have created a new generation configuration named `Web`.



Procedure 7.2. Webdoclet Configuration

1. Select the `Web` configuration.
2. In the lower-left area, right-click to popup the menu and choose `Add Doclet`.
3. A list of available doclets will appear. Choose `webdoclet` and click `OK`.
4. On the lower-right area, you see the properties of the `ejbdoclet`.
   - Set the `destDir` property to `src/WEB-INF`.

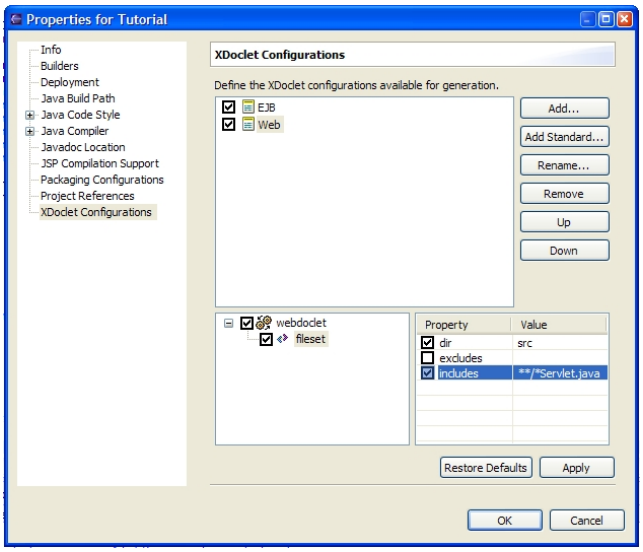
Our configuration now contains a `webdoclet` that will produce files in the `src/WEB-INF` folder.



Procedure 7.3. Fileset Configuration

1. In the lower-left area, right-click on `webdoclet` to popup the menu and choose `Add`.
2. A list of available subtasks will appear. Choose `fileset` and click `Ok`.
3. On the lower-right area, you see the properties of the fileset.
   a. Set the `dir` property to `src`.
   b. Uncheck `excludes`
   c. Set the `includes` property to `**/*Servlet.java`.

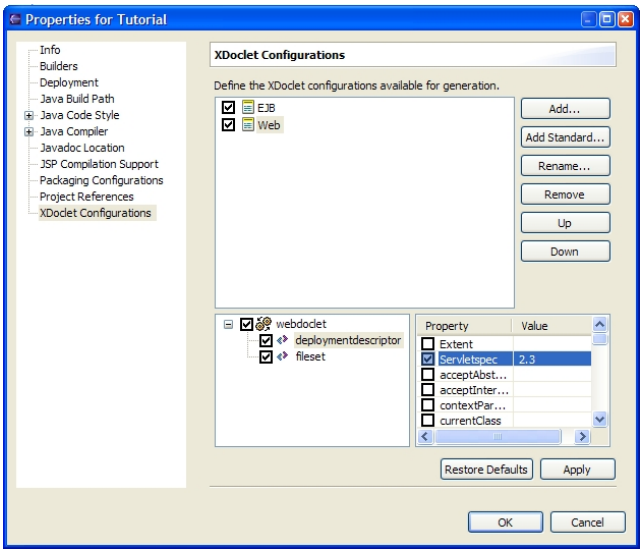
Our configuration now contains a `webdoclet` with a `fileset` that contains the `src` directory, and all files under it that end in `Servlet.java`.



Procedure 7.4. Deployment Descriptor Configuration

- Add a new `deploymentdescriptor` subtask to the `webdoclet` (see above).
  - Set the `Servletspec` property to `2.3`.

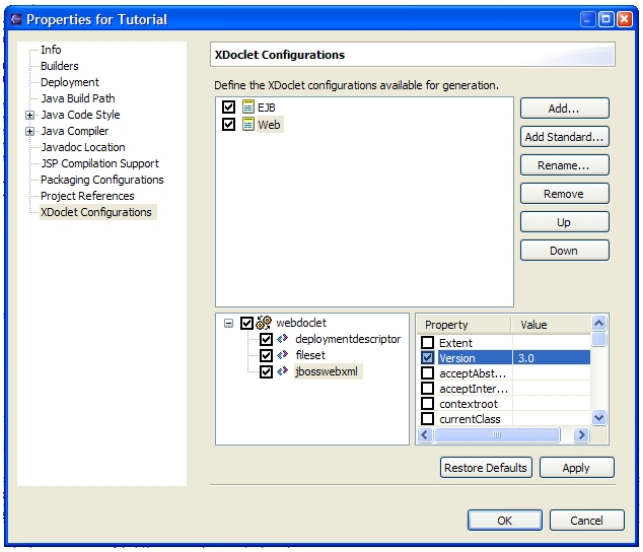
All of the standard Web deployment descriptors will now be placed in the `src/WEB-INF` directory (property is inherited from `webdoclet`).



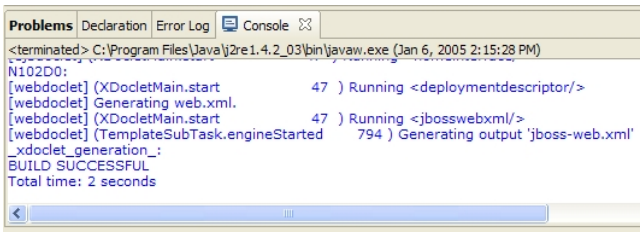
Procedure 7.5. JBoss Configuration

- Add a new `jbosswebxml` subtask to the `webdoclet` (see above).
    - Set the `Version` property to `3.0`.

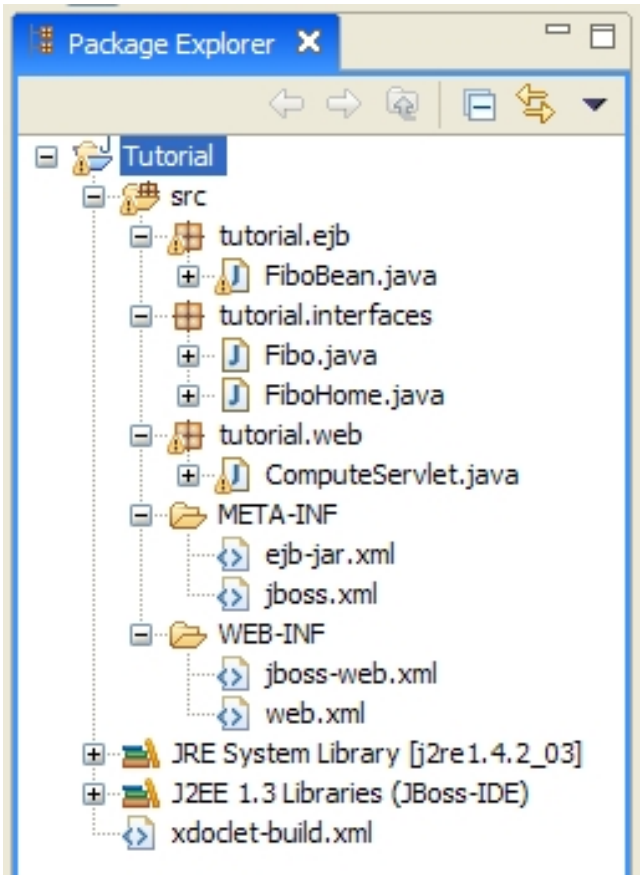
All of the JBoss-specific Web deployment descriptors will now be placed in the `src/WEB-INF` directory (property is inherited from `webdoclet`).



Click `OK` and the XDoclet configuration for the `Tutorial` will be saved. Once the configuration is saved, right-click on the `Tutorial` project and select `Run XDoclet`. The XDoclet generation will display its output in the console. The output should look like this:



After the generation, you should have a project that looks like this. Note that a `WEB-INF` folder has been created with the web deployment descriptors (both standard and jboss).

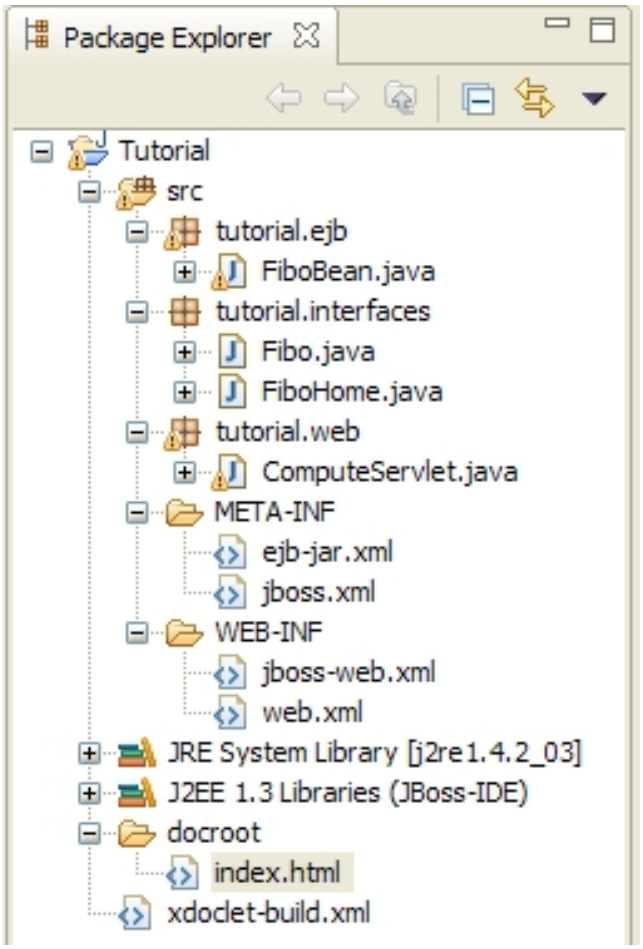


In order to run our servlet, we'll need to create an HTML page that passes it parameters.

Procedure 7.6. Creating the HTML Page

1. Create a `docroot` folder under the root of the project.
2. Create an empty file named `index.html` under the `docroot` folder.

The `index.html` file is intended to be the default page for the Web application and contains a form that will be posted to the Servlet.



The following content should be copied into the `index.html` file:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
   <head>
      <title>
         Fibonacci Application
      </title>
   </head>
   <body>
      <h1>Fibonacci Form</h1>
      <form action="Compute" method="POST" >
         <table cellspacing="2" cellpadding="2" border="0">
            <tr>
               <td>
                  Limit :
               </td>
               <td>
                  <input type="text" name="limit" value="50">
               </td>
            </tr>
            <tr>
               <td>
                  <input type="submit" name="Compute" value="Compute">
               </td>
               <td>
                  <input type="Reset">
               </td>
```

```
            </tr>
         </table>
      </form>
   </body>
</html>
```

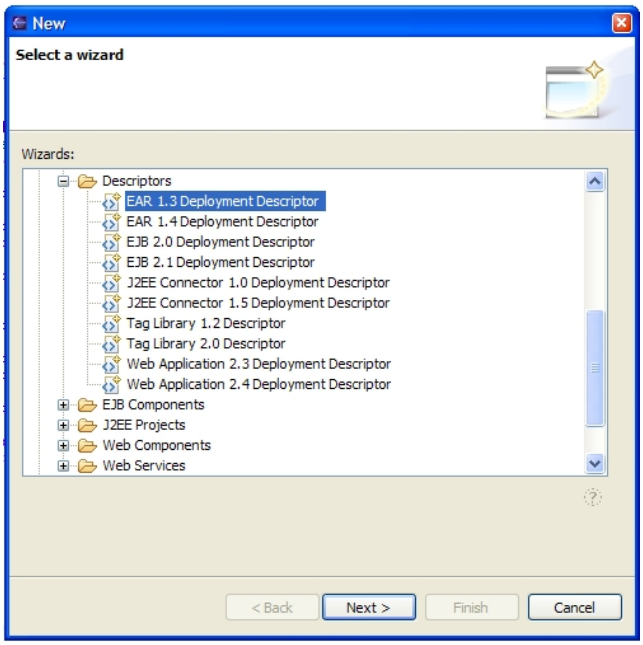
8

The J2EE Application

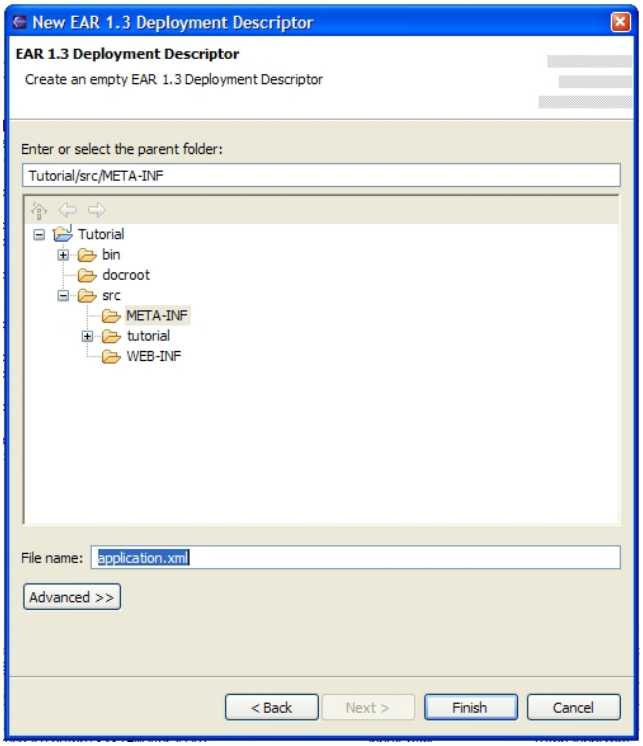
This project is intended to be a complete J2EE application. We are going to create some additional files to have all the materials needed to build it.

Procedure 8.1. Creating the application.xml

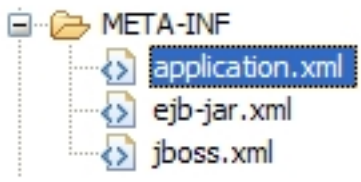
1. Right click on the `src/META-INF` folder, and choose `New` > `Other...`.
2. Choose `JBoss-IDE` > `Descriptors` > `EAR 1.3 Deployment Descriptor`, and click `Next`.



Make sure `application.xml` is the name of the file, and click `Finish`



Your `META-INF` directory should now look like this:



Now double click on the `application.xml` to open it, and make sure the content looks like this (most of the content is already there for you):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application PUBLIC
        "-//Sun Microsystems, Inc.//DTD J2EE Application 1.3//EN"
        "http://java.sun.com/dtd/application_1_3.dtd">
<application>
        <display-name>Sum Application</display-name>
        <module>
                <ejb>FiboEJB.jar</ejb>
        </module>
        <module>
                <web>
                        <web-uri>FiboWeb.war</web-uri>
                        <context-root>/fibo</context-root>
                </web>
        </module>
</application>
```

9

The Packaging

JBoss-IDE provides an easy way to configure the packaging of various archives. There is no restriction of what can be packaged. In this tutorial, four packaging configurations will be defined:

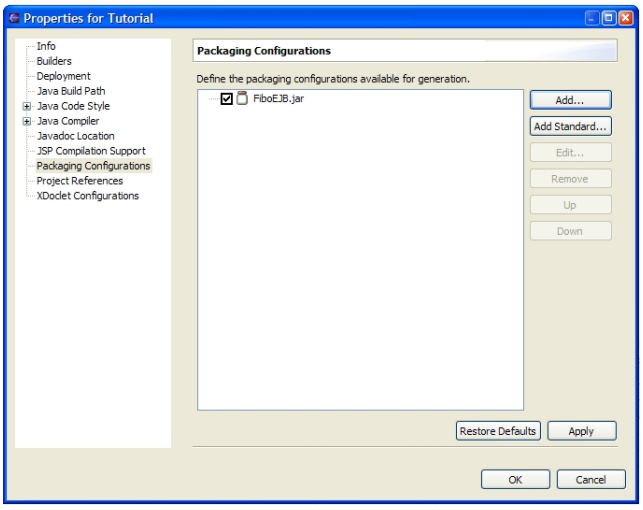
- The EJB JAR. It will contain the EJB classes and interfaces, as well as the ejb-jar.xml and jboss.xml deployment descriptors.
- The EJB Client JAR. It will contain the EJB interfaces.
- The Web Application WAR. It will contain the Servlet class, the EJB client Jar, as well as the web.xml deployment descriptors.
- The J2EE Application EAR. It will contain the EJB Jar and the Web Application War, as well as the application.xml deployment descriptor.

When launched, these four packaging configurations will create the J2EE application ready to be deployed.

9.1. FiboEJB.jar creation

Procedure 9.1. Creating the EJB JAR

1. Edit the project properties by right clicking on the project and select `Properties`.
2. In the property page, select `Packaging Configurations`.
3. Right-click in the area to pop-up the menu and choose `Add Archive`. Type `FiboEJB.jar` in the dialog and click `OK`.
4. You have created a new packaging configuration that will produce the `FiboEJB.jar` file.



We want to add the EJB classes and interfaces. Eclipse has generated the compiled classes into the `bin` folder (declared as the default output dir of the project).

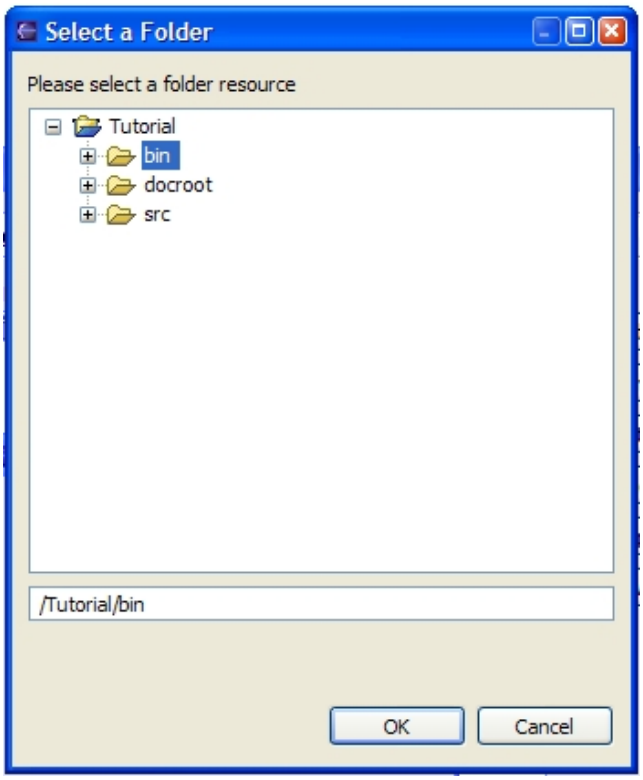
Select the `FiboEJB.jar` item and right-click in the area to pop-up the menu and choose `Add Folder`. A "Folder Selection" dialog appears.

This dialog allows to select which folder (local to workspace or in the file system) to include into the package, to specify include and exclude filters (A la Ant) and to set a prefix that will be append when building the package.

Click on `Project Folder`. A "Folder Chooser" dialog appears.

This dialog allows selecting which folder to include. This folder can be choosen among all the opened projects.

Select the `/Tutorial/bin` folder and click `OK`.

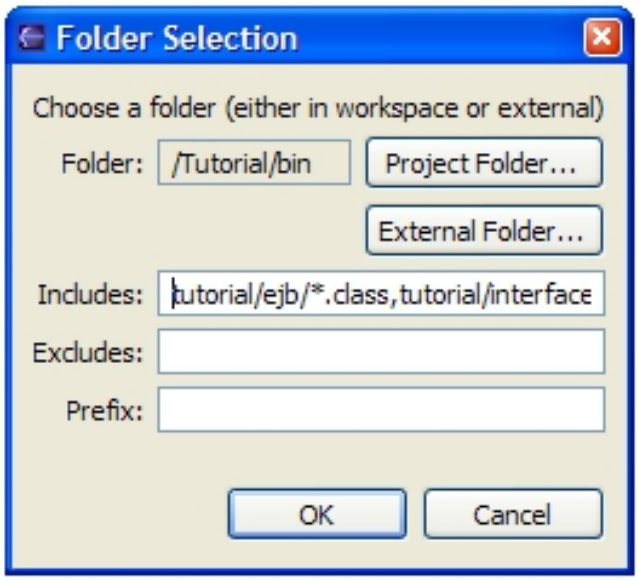


The folder is now `/Tutorial/bin`.

As we only want the EJB classes and interfaces, specify the following as an include filter:

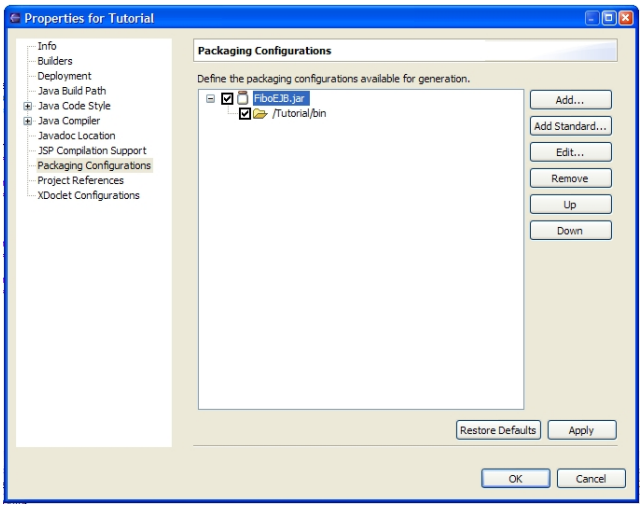
```
tutorial/
ejb/*.class,tutorial/interfaces/*.class
```

Click on `OK`

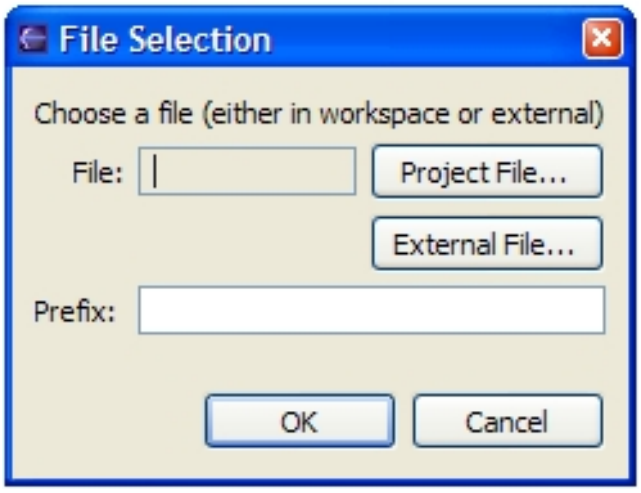


We now want to add the standard EJB deployment descriptor.

Select the `FiboEJB.jar` item and right-click in the area to pop-up the menu and choose `Add File`. A "File Selection" dialog appears.



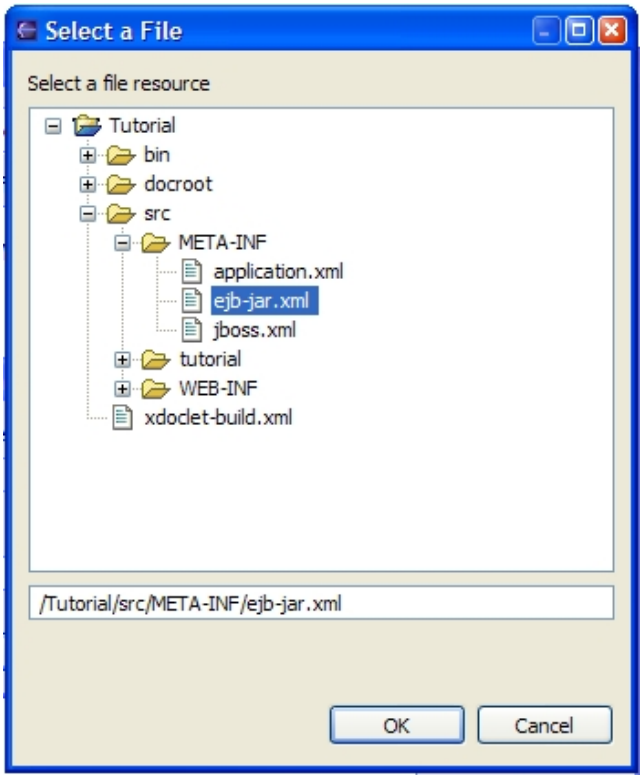
This dialog allows you to select which file (local to workspace or in the file system) to include in the package and to set a prefix which will be appended when building the package.



Click on `Project File`. A “File Chooser” dialog appears.

This dialog allows to select which file to include. This file can be choosen among all the opened projects.

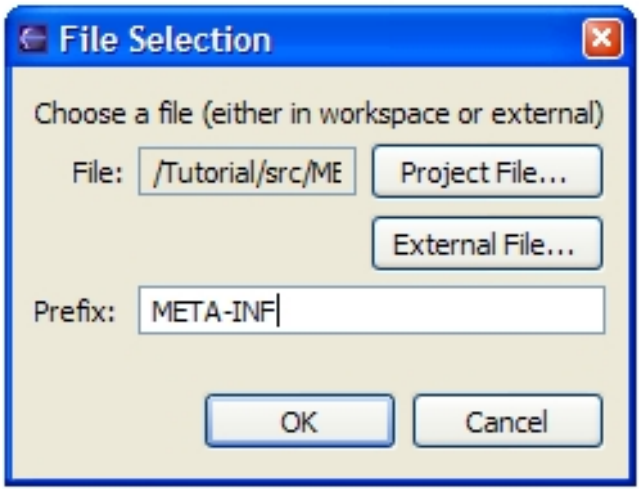
Select the `/Tutorial/src/META-INF/ejb-jar.xml` folder and click `OK`.



The file is now `/Tutorial/src/META-INF/ejb-jar.xml`.

The `ejb-jar.xml` must be located under the `META-INF` directory of the EJB package. Set the prefix to `META-INF`.

Click on `OK`.

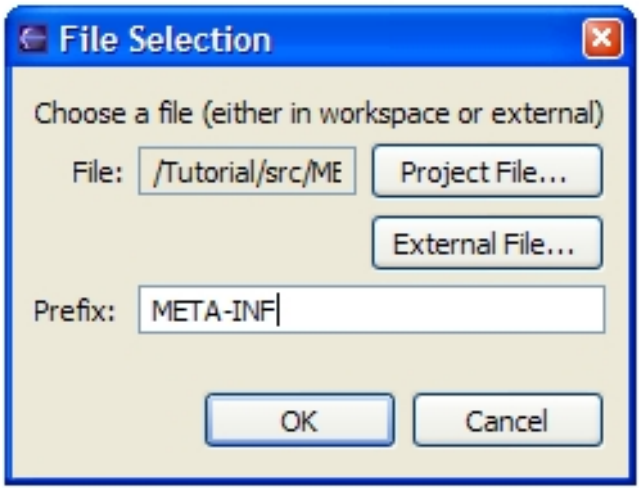


To add the specific EJB deployment descriptor, select the `FiboEJB.jar` item and right-click in the area to pop-up the menu and choose `Add File`.

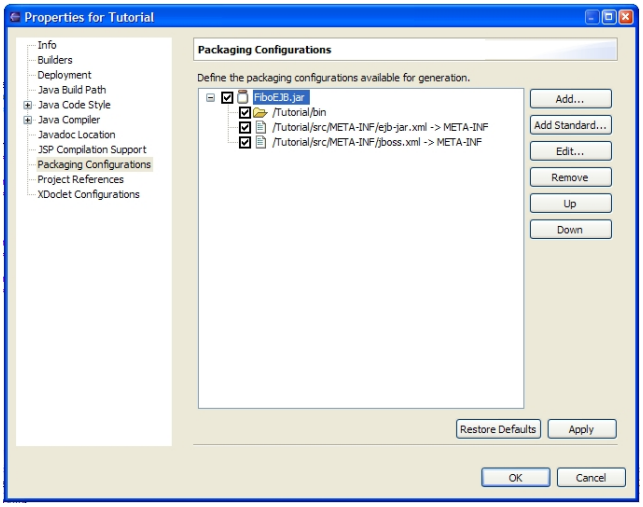
The file to choose is `/Tutorial/src/META-INF/jboss.xml`.

The `jboss.xml` must be located under the `META-INF` directory of the EJB package. Set the prefix to `META-INF`.

Click on `OK`.



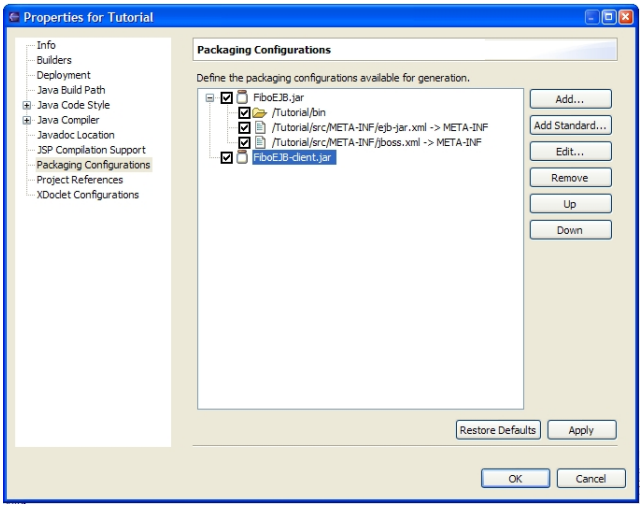
The packaging configuration for the `FiboEJB.jar` is now complete.



9.2. FiboEJB-client.jar creation

Click the `Add` button on the right side of the list. Type `FiboEJB-client.jar` in the dialog and click `OK`.

You have created a new packaging configuration that will produce the `FiboEJB-client.jar` file.



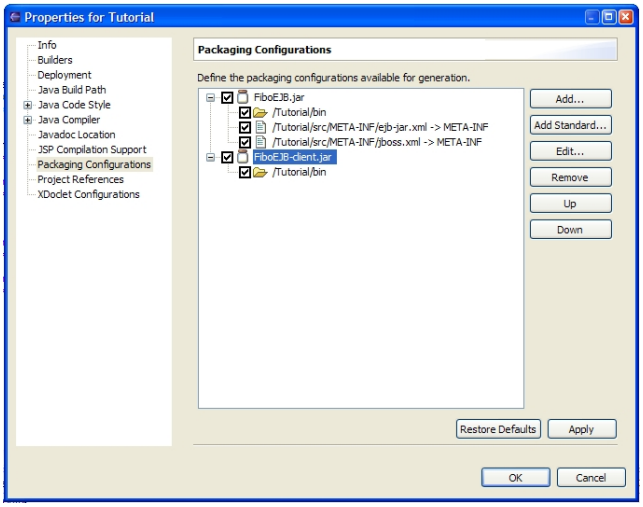
Select the `FiboEJB-client.jar` item and right-click in the area to pop-up the menu and choose `Add Folder`. A “Folder Selection” dialog appears.

Click on `Project Folder` and select the `/Tutorial/bin` folder from the “Folder Chooser” dialog.

As we only want the EJB interfaces, set the include filter to `tutorial/interfaces/*.class`.

Click on `OK`.

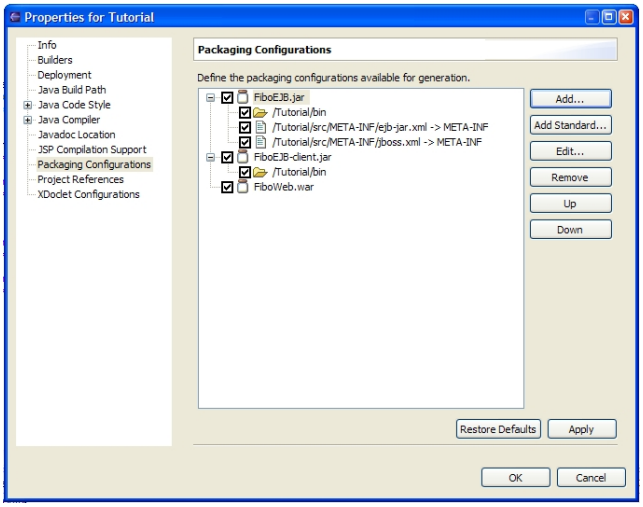
The packaging configuration for the `FiboEJB-client.jar` is now complete.



9.3. FiboWeb.war creation

Click the `Add` button on the right side of the list. Type `FiboWeb.war` in the dialog and click `OK`.

You have created a new packaging configuration that will produce the `FiboWeb.war` file.

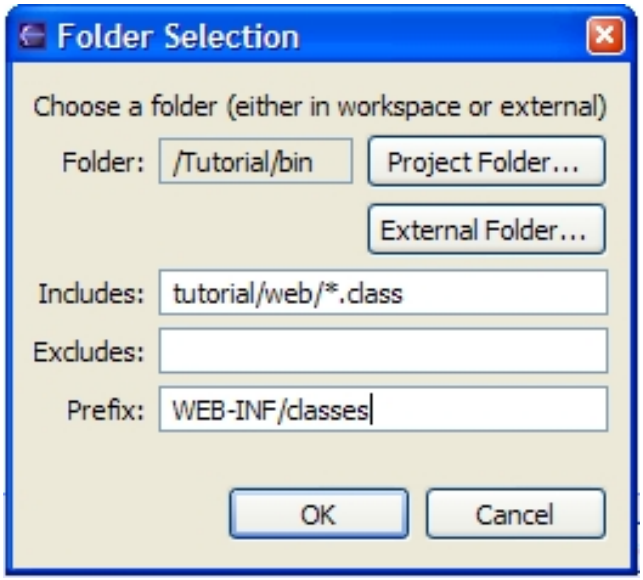


Select the `FiboWeb.war` item and right-click in the area to pop-up the menu and choose `Add Folder`. A "Folder Selection" dialog appears.

Click on `Project Folder` and select the `/Tutorial/bin` folder from the "Folder Chooser" dialog.

As we only want the Servlet class, set the include filter to `tutorial/web/*.class`.

The classes must be located under the `WEB-INF/classes` of the War package. Set the prefix to `WEB-INF/classes`.



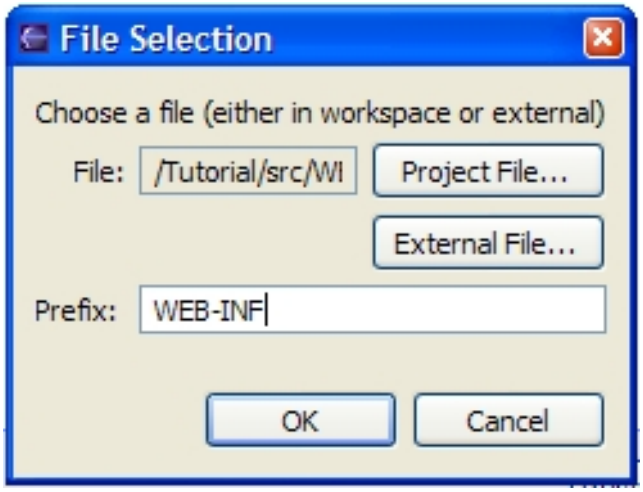
Click on `OK`.

To add the standard Web deployment descriptor, select the `FiboWeb.war` item and right-click in the area to pop-up the menu and choose `Add File`. A “File Selection” dialog appears.

The file to choose is `/Tutorial/src/WEB-INF/web.xml`.

The `web.xml` must be located under the `WEB-INF` of the War package. Set the prefix to `WEB-INF`.

Click on `OK`.

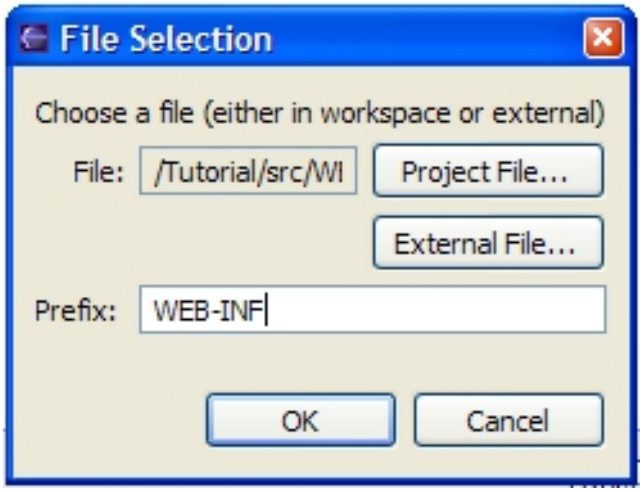


To add the JBoss specific Web deployment descriptor, select the `FiboWeb.war` item and right-click in the area to pop-up the menu and choose `Add File`. A “File Selection” dialog appears.

The file to choose is `/Tutorial/src/WEB-INF/jboss-web.xml`.

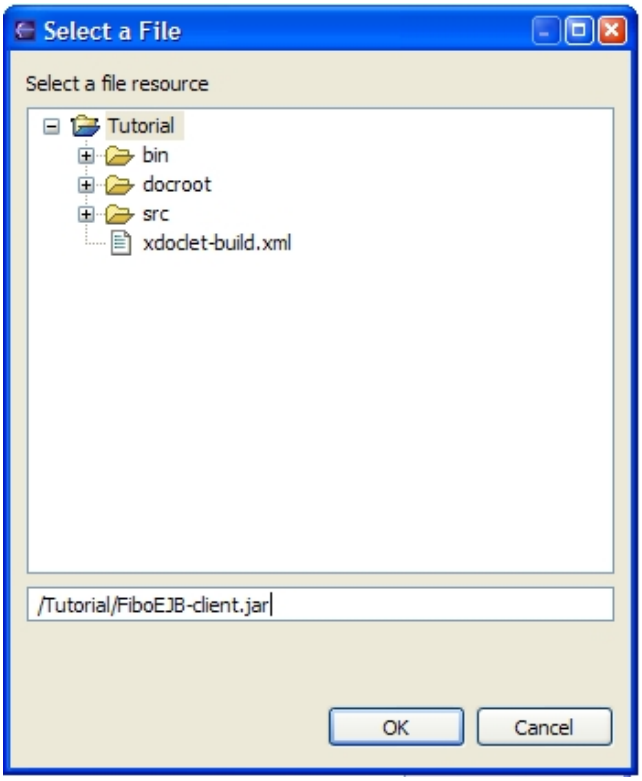
The `jboss-web.xml` must be located under the `WEB-INF` of the War package. Set the prefix to `WEB-INF`.

Click on `OK`.



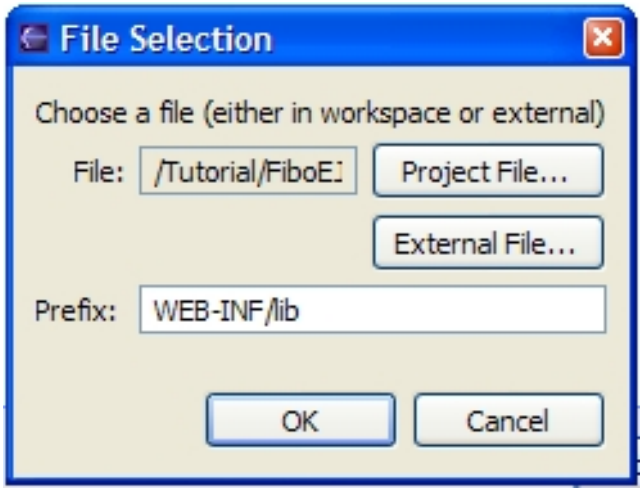
To add the EJB Client Jar, select the `FiboWeb.war` item and right-click in the area to pop-up the menu and choose `Add File`. A “File Selection” dialog appears.

The file to choose is `/Tutorial/FiboEJB-client.jar`. But it doesn’t exist yet as the packaging has not been run. Instead of selecting it, go in the text field and type the name of the file `/Tutorial/FiboEJB-client.jar`. Even if the file doesn’t exist, it can be added to a packaging configuration.



The `FiboEJB-client.jar` must be located under the `WEB-INF/lib` directory of the War package. Set the prefix to `WEB-INF/lib`.

Click on `OK`.

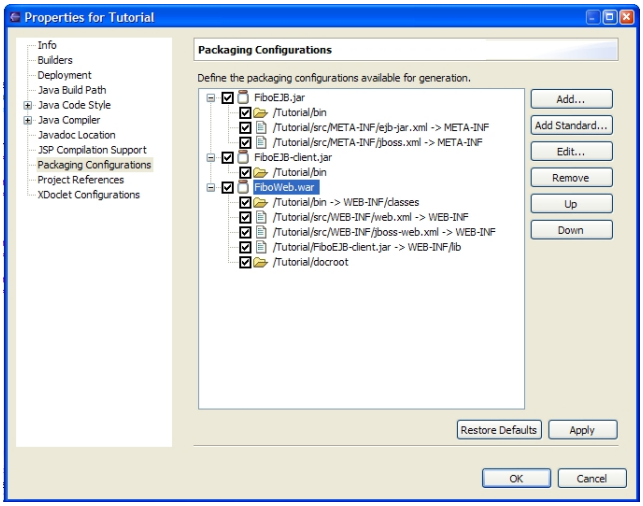


Select the `FiboWeb.war` item and right-click in the area to pop-up the menu and choose `Add Folder`. A “Folder Selection” dialog appears.

Click on `Project Folder` and select the `/Tutorial/docroot` folder from the “Folder Chooser” dialog. This is the content of the Web Application.

Click on `OK`.

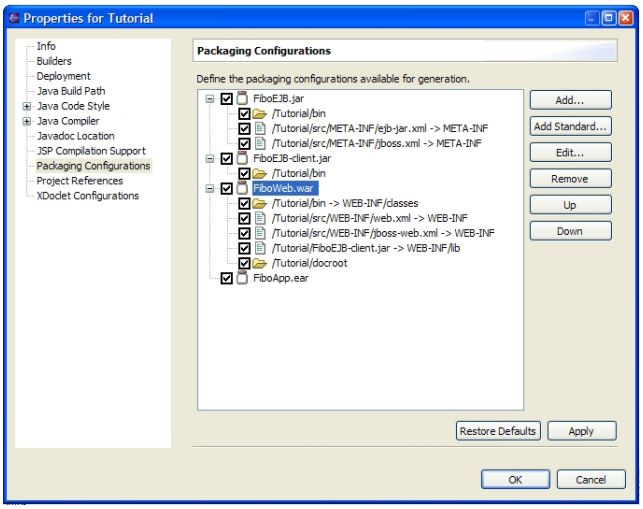
The packaging configuration for the `FiboWeb.war` is now complete.



9.4. FiboApp.ear creation

Click the `Add` button on the right side of the list. Type `FiboApp.ear` in the dialog and click `OK`.

You have created a new packaging configuration that will produce the `FiboApp.ear` file.

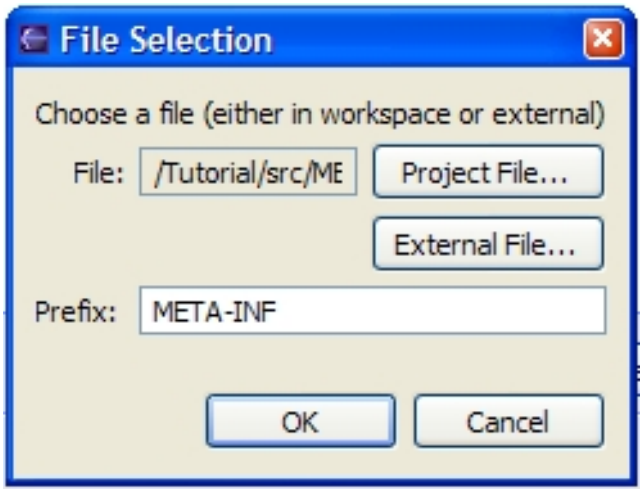


To add the application deployment descriptor, select the `FiboApp.ear` item and right-click in the area to pop-up the menu and choose `Add File`. A “File Selection” dialog appears.

The file to choose is `/Tutorial/src/META-INF/application.xml`.

The `application.xml` must be located under the `META -INF` of the EAR package. Set the prefix to `META -INF`.

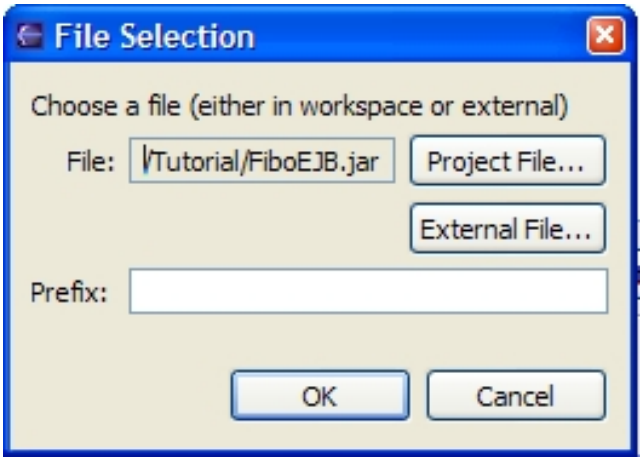
Click on `OK`.



To add the EJB module, select the `FiboApp.ear` item and right-click in the area to pop-up the menu and choose `Add File`. A “File Selection” dialog appears.

The file to choose is `/Tutorial/FiboEJB.jar`. But it doesn’t exist yet as the packaging has not been run. Instead of selecting it, go in the text field and type the name of the file `/Tutorial/FiboEJB.jar`. Even if the file doesn’t exist, it can be added to a packaging configuration.

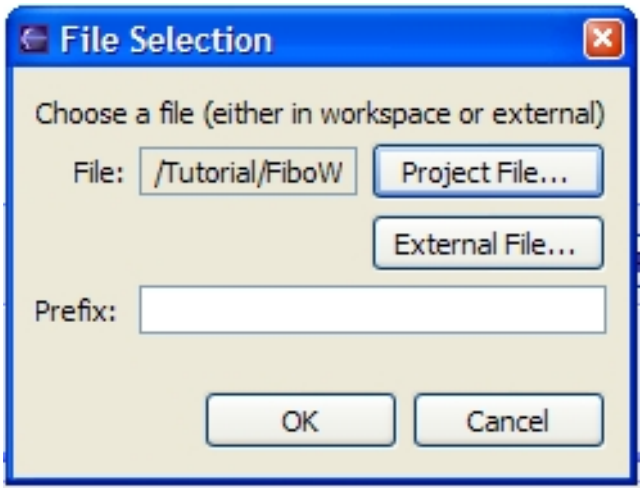
Click on `OK`.



To add the Webmodule, select the `FiboApp.ear` item and right-click in the area to pop-up the menu and choose `Add File`. A “File Selection” dialog appears.

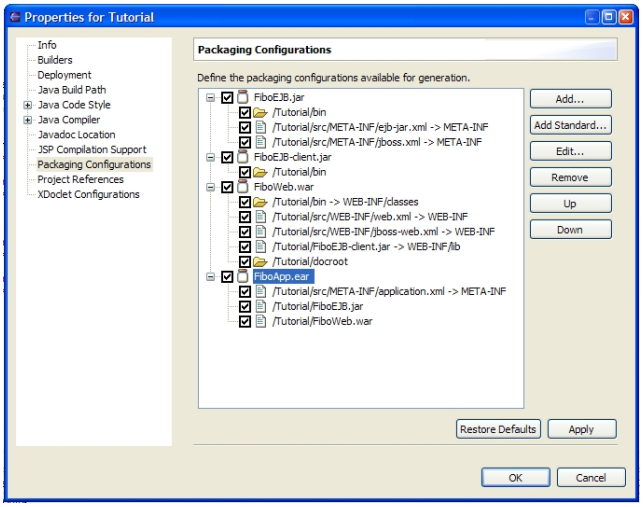
The file to choose is `/Tutorial/FiboWeb.war`. But it doesn’t exist yet as the packaging has not been run. Instead of selecting it, go in the text field and type the name of the file `/Tutorial/ FiboWeb.war`. Even if the file doesn’t exist, it can be added to a packaging configuration.

Click on `OK`.

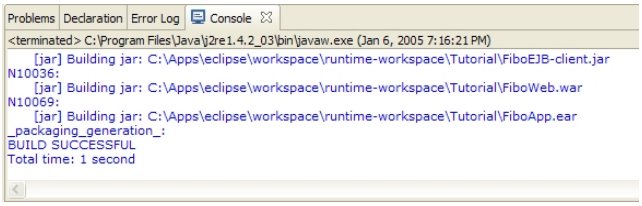


The packaging configuration for the `FiboApp.ear` is now complete.

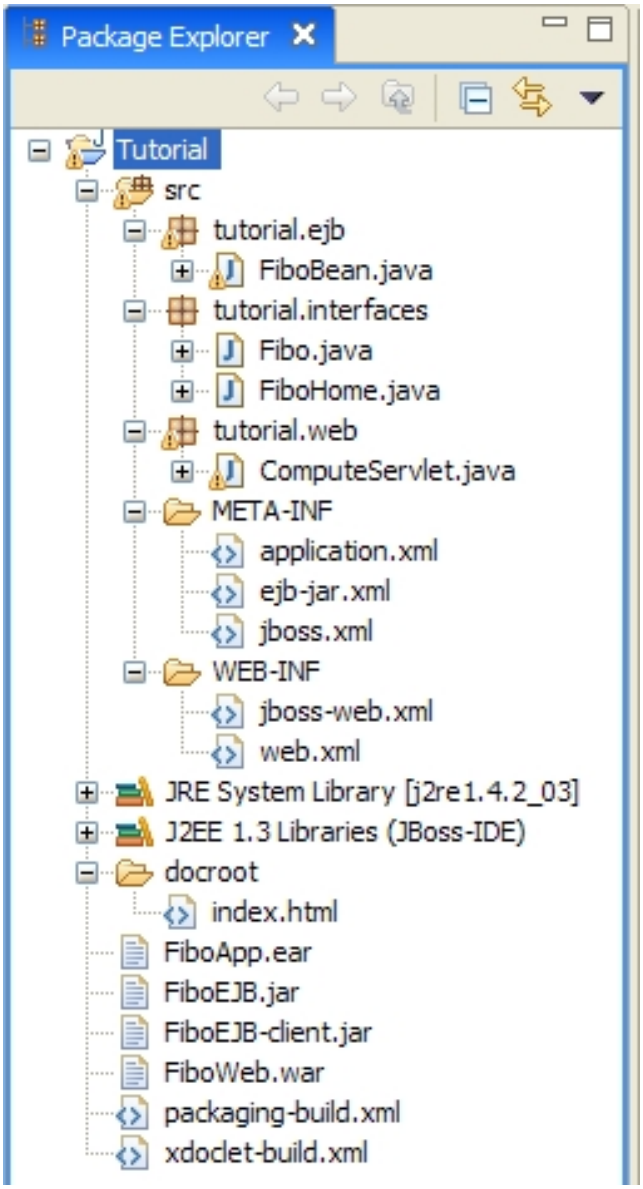
Click `OK` to save the packaging configurations.



Right-click on the project and select `Run Packaging`. The packaging will display its output in the console. The output should look like this:



After the execution, you should have a project that looks like this:

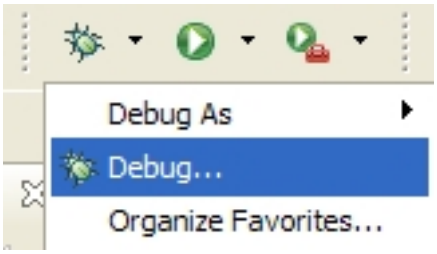


10

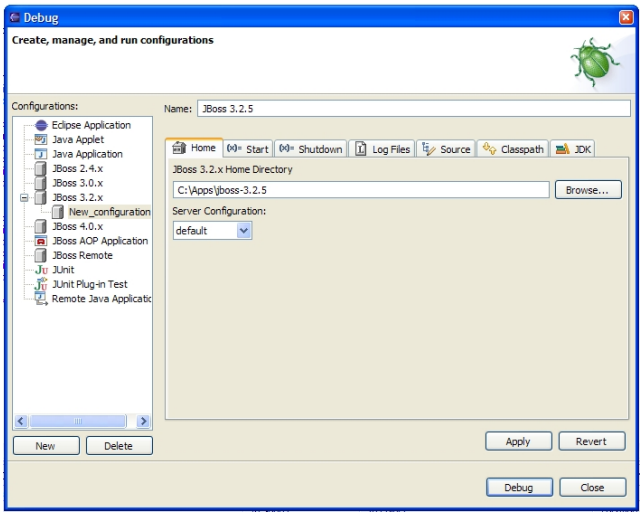
JBoss Configuration and Launch

Now, it is time to configure the JBoss server if it has not been done yet.

Click on the debug shortcut and select `Debug...` to open the debug configurations.



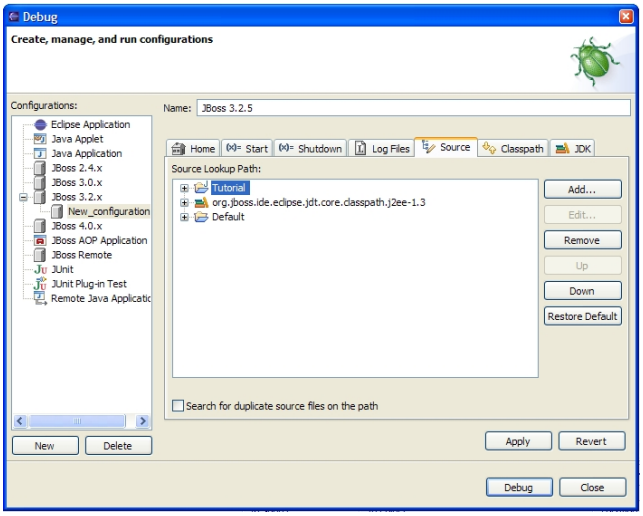
The debug dialog allows you to configure the available JBoss configurations that will be used for debugging.



In order to view source code when debugging, you must include the project in the source lookup path; otherwise Eclipse will complain that it cannot locate the source.

To specifiy a source lookup path, go into the JBoss launch configuration and select the `Source` tab. Click on the `Add` button and select `Java Project`. Select your project and click `OK`.

NB : Older version of JBoss-IDE added all opened projects to the source lookup path, but it lead to source conflicts and was removed.



Select the configuration you want to launch and click on `Debug` and you will see JBoss starting. The output is sent to the console.

11

Deployment

The deployment within JBoss-IDE can be done in two ways:

- A file-system copy: Copies a file from your project into any other location on your computer (including network drives, etc)
- A local deployment through the MainDeployer MBean (Experimental). The URL of the resource is sent to the MainDeployer Mbean, which deploys and watches it.

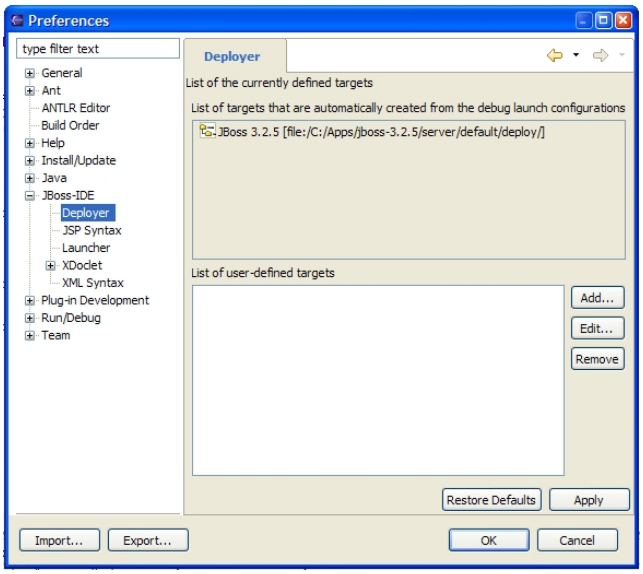
In addition, the deployment target is stored during the workbench session. This means that if you have deployed a package on a target, you can redeploy or undeploy it without specifying the target. The Deployer plugin automatically creates file-system targets from the debug configurations. Other deployment target can be defined.

Select `Window` > `Preferences`. The workbench preferences appears.

Select `JBoss-IDE` > `Deployer` to display the defined deployment targets.

The upper area contains the file system targets build upon the debug configuration defined.

The lower area contains the user-defined deployment targets.

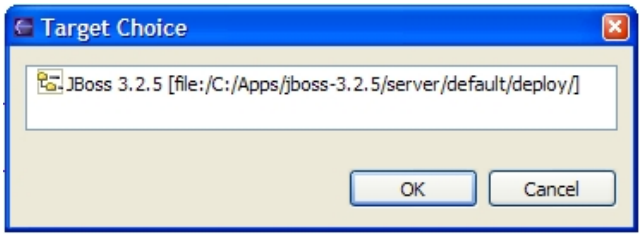


We assume that we want to deploy to a pre-defined JBoss instance and we don't define custom deployment targets.

The deployment is fairly simple. Right click on the `FiboApp.ear` file and select the `Deployment` > `Deploy To...` item.

A dialog box appears with the list of the deployment targets. It contains both the default and the user-defined deployment targets.

Select the one you are interested in (the one for the running server is probably a good idea !).



In the console view, you should see some deployment activity. The J2EE application is now deployed.

When a resource is deployed, a small decorator appears in the top-left corner of the icon.

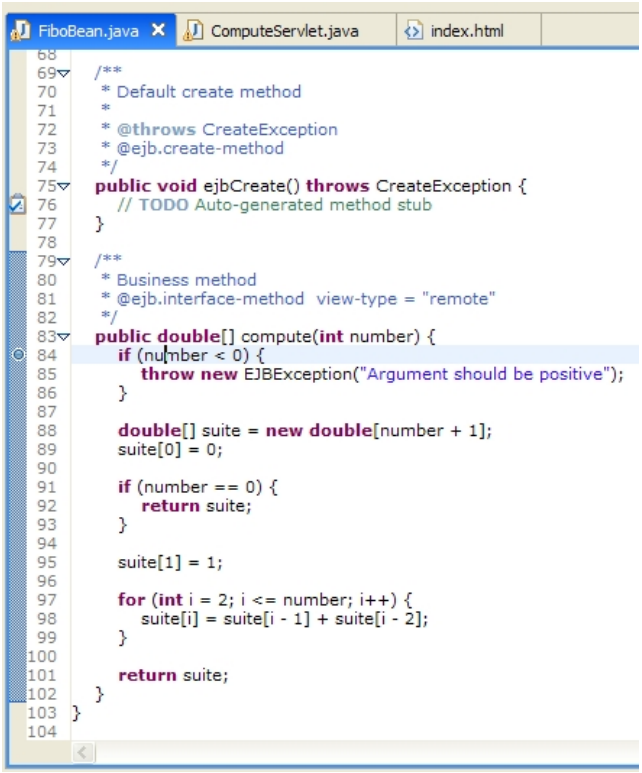
12

Debugging

Prior to the debugging, we need to set some breakpoints inside the code.

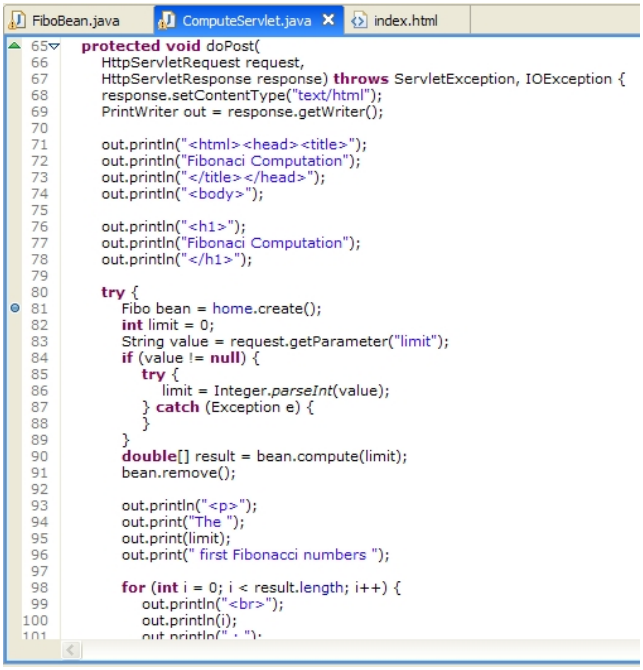
Open the `FiboBean.java` file. Double click in left column to create a breakpoint.

In the example, the breakpoint is set in front of the test.

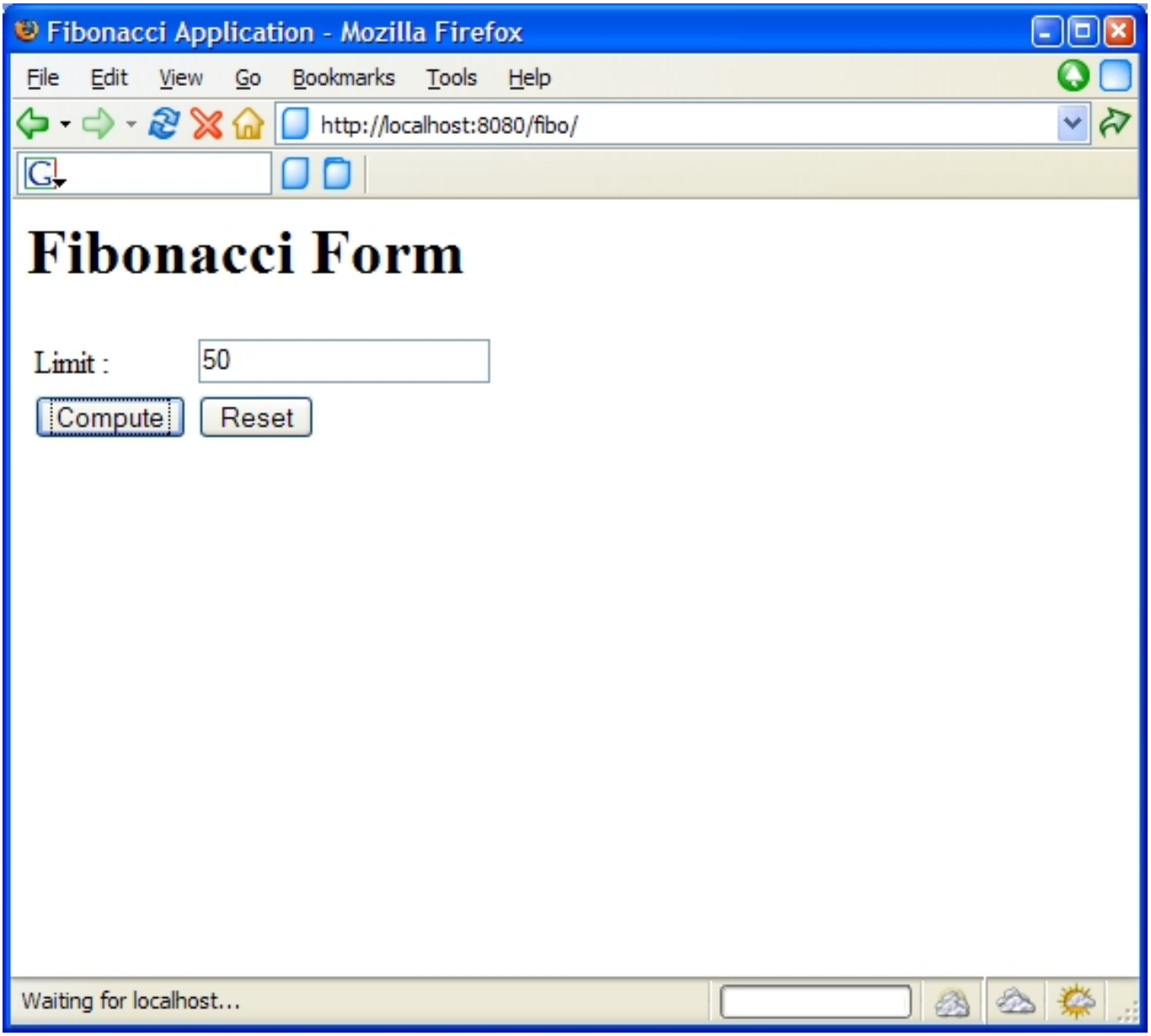


Open the `ComputeServlet.java` file. Double click in left column to create a breakpoint.

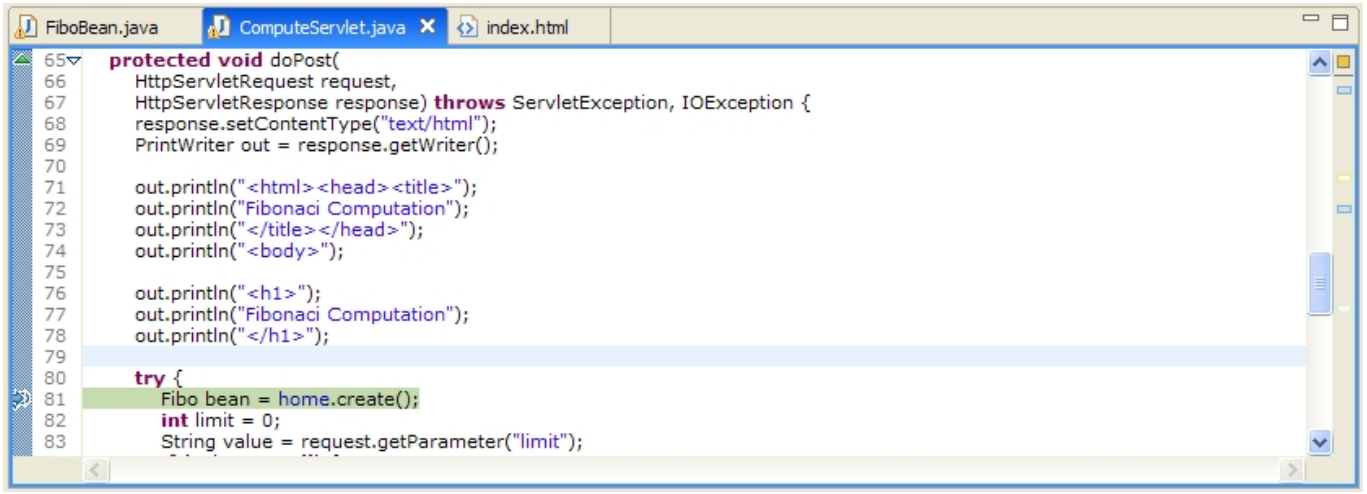
In the example, the breakpoint is set in front of the EJB creation.



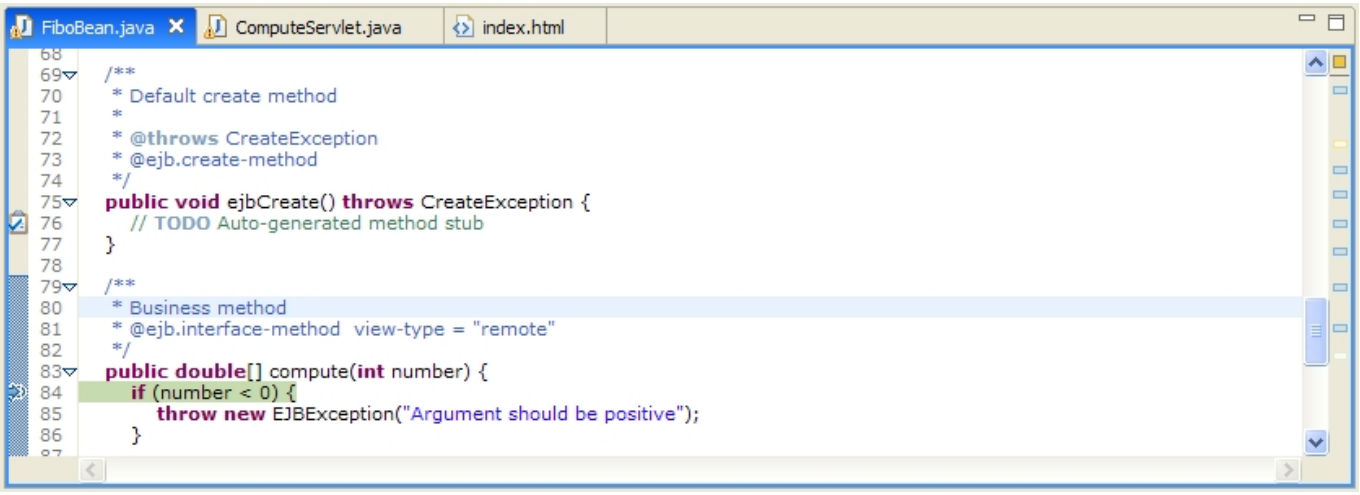
Open a web browser and type http://localhost:8080/fibo/. The host/port can change if the web server listens on another host/port. You should see a simple form like the one above. Enter a positive value in the field and press `Compute`.



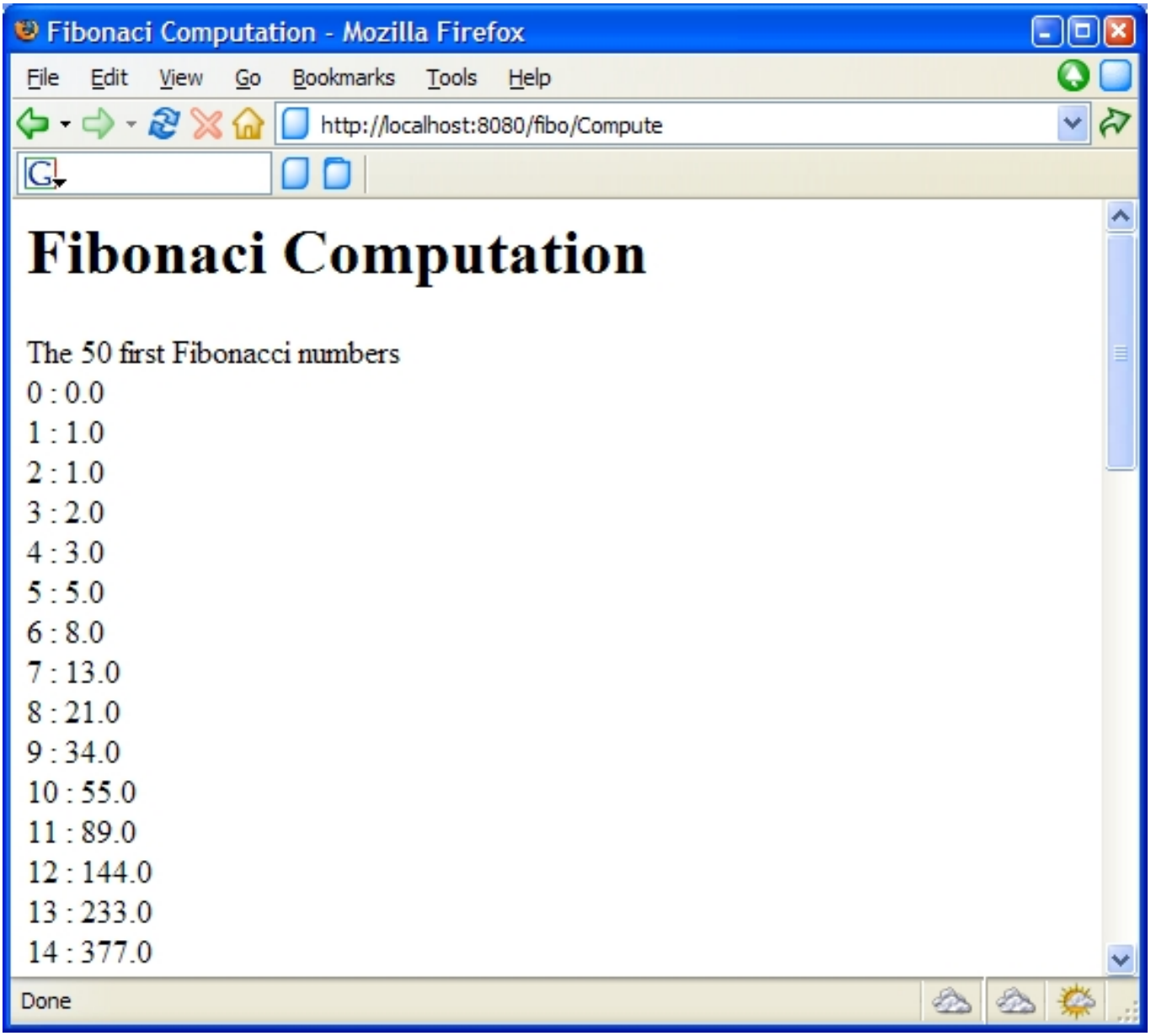
Switch to your Eclipse workbench. You should see that execution has been suspended on the first breakpoint (in the servlet). You can go step by step in the code or continue with execution.



Another suspension occurs when hitting the second breakpoint (in the EJB). You can go step by step in the code or continue with execution.



After resuming execution, the response should be in the browser. It should look something like this:


Fibonaci Computation - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/fibo/Compute
Fibonaci Computation
The 50 first Fibonacci numbers
0 : 0.0
1 : 1.0
2 : 1.0
3 : 2.0
4 : 3.0
5 : 5.0
6 : 8.0
7 : 13.0
8 : 21.0
9 : 34.0
10 : 55.0
11 : 89.0
12 : 144.0
13 : 233.0
14 : 377.0
Done

13

Conclusion

This simple tutorial was intended to give an overview of what is possible with JBoss-IDE. We hope that it will be useful for developers who want to develop for JBoss in Eclipse