



University of New York Tirana
Faculty of Engineering and Architecture
Rruga e Kavajës, pranë 21 Dhjetorit (Sheshi Atatürk)
Tirane, Shqipëri

Master of Science in Computer Science

**Distributed Systems
Manual for Laboratory Practice**

Enterprise JavaBeans

PART II

**Introducing Servlets and Facelets
Connecting Servlets and Facelets with JavaBeans
Connecting Servlets to Enterprise JavaBeans
Connecting Enterprise JavaBeans to Databases
A Web Banking Application with EJB and MySQL**

Marenglen Biba, Ph.D
Department of Computer Science,
E-mail : marenglenbiba@unyt.edu.al

1. Document Purpose

For running the programs, a correct configuration of the running environment is necessary (path and classpath variables).

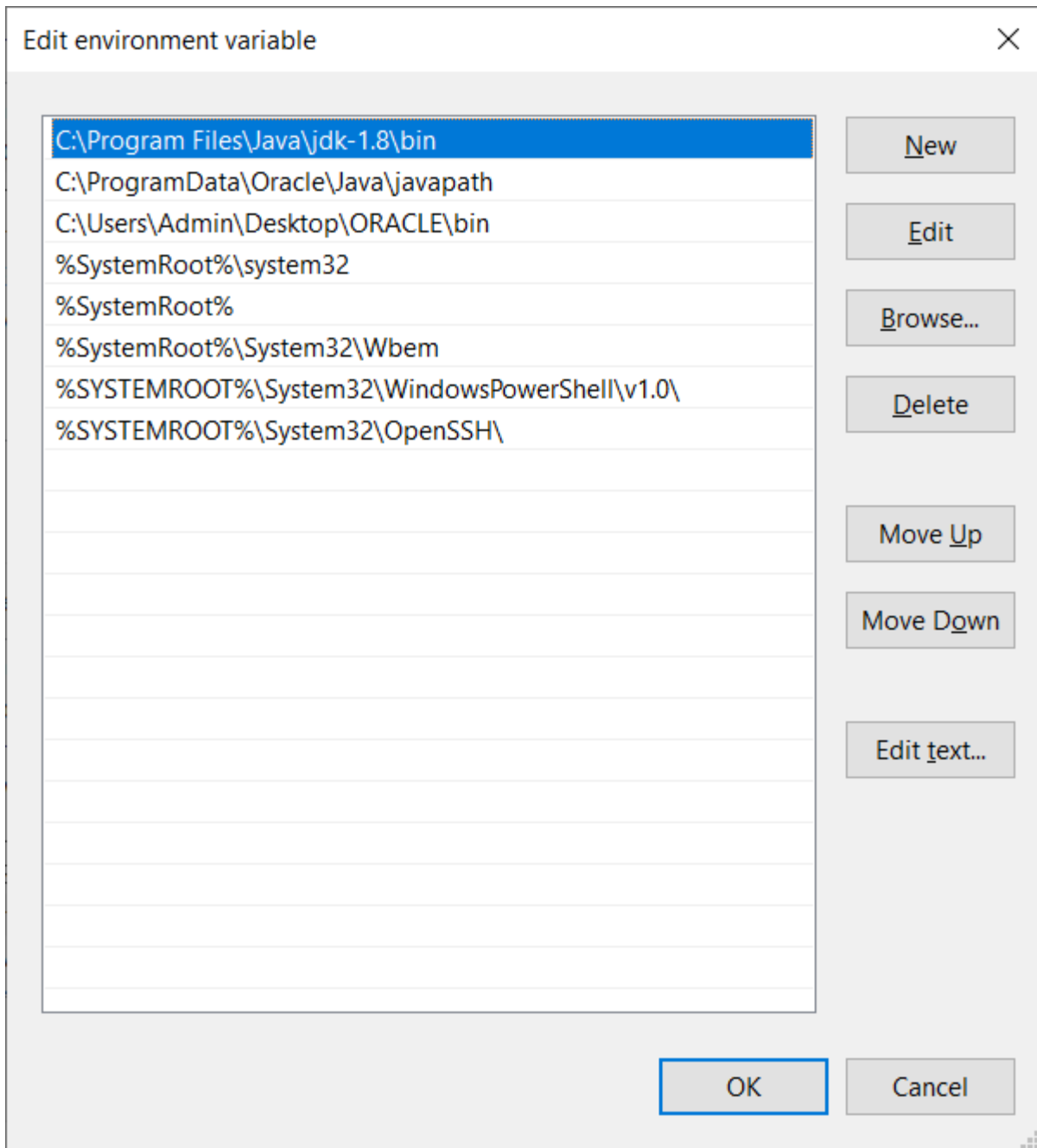
- Install Java SE (JDK) JDK8
<https://www.oracle.com/java/technologies/javase/javase8-archive-downloads.html>
- Install Java EE JDK7
<http://www.oracle.com/technetwork/java/javaee/downloads/java-ee-sdk-7-downloads-1956236.html>
- Install Netbeans 8.2
<https://dlc-cdn.sun.com/netbeans/8.2/final/?pagelang=>
- Install MySQL 5.0 and MySQL WorkBench 8.0

Set path and Path variables in the operating system

Click on Environment Variables.

Find the Path system variable and click Edit. Set the value of the variable to the directory where you have installed Java, for example:

D:\Program Files\Java\jdk1.8.0\bin



Ensure that the required JDK software is installed on your system and that the `JAVA_HOME` environment variable points to the JDK installation directory, not the Java Runtime Environment (JRE) software.

Environment Variables



User variables for Admin

Variable	Value
MOZ_PLUGIN_PATH	C:\Program Files (x86)\Foxit Software\Foxit PDF Reader\plugi...
OneDrive	C:\Users\Admin\OneDrive
Path	C:\Users\Admin\AppData\Local\Programs\Python\Python312...
TEMP	C:\Users\Admin\AppData\Local\Temp
TMP	C:\Users\Admin\AppData\Local\Temp

New... Edit... Delete

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
JAVA_HOME	C:\Program Files\Java\jdk-1.8
NUMBER_OF_PROCESSORS	8
OS	Windows_NT
Path	C:\Program Files\Java\jdk-1.8\bin;C:\ProgramData\Oracle\Jav...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64

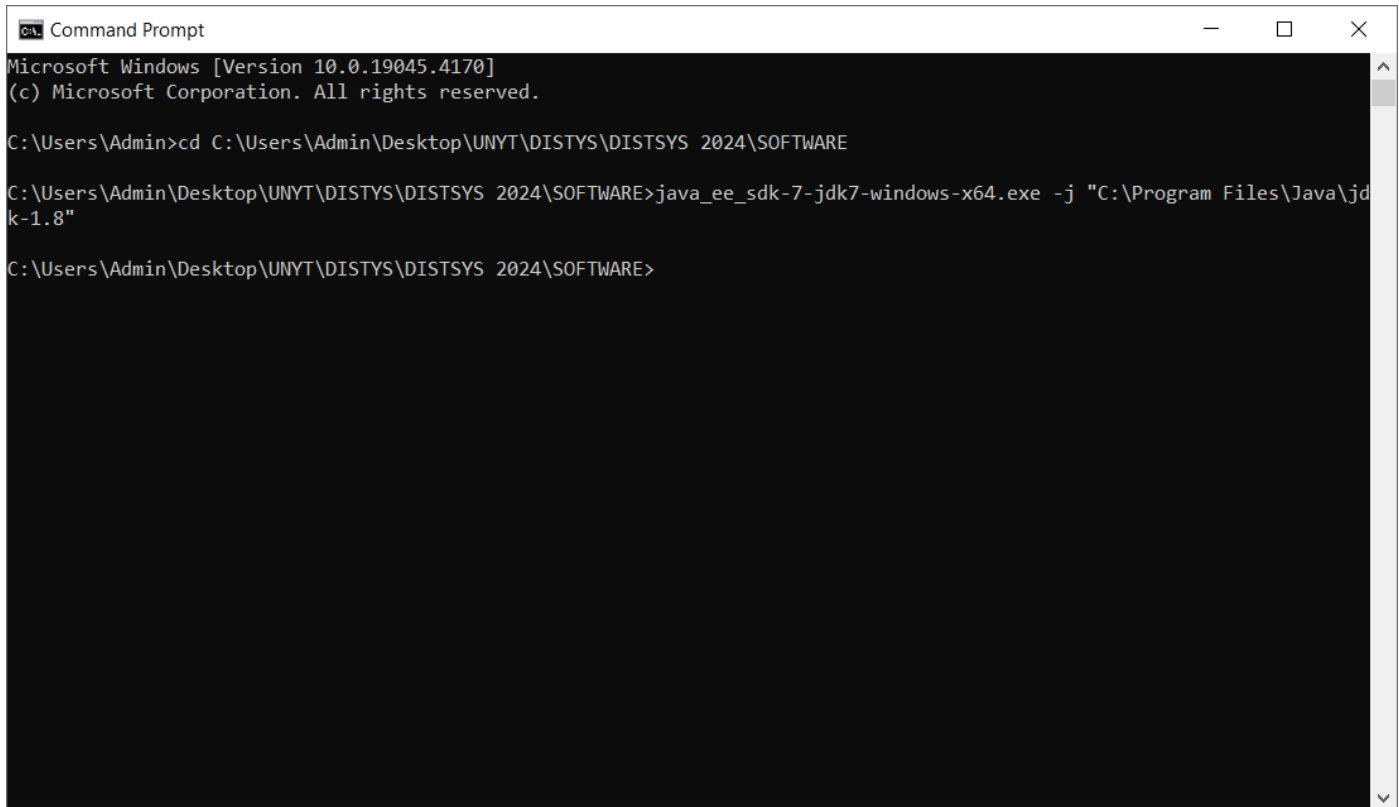
New... Edjt... Delete

OK Cancel

Download and install the Netbeans IDE by double clicking the executable installation file.

Download and install Java EE SDK.

If the .exe file does not start use the following command:



```
Command Prompt
Microsoft Windows [Version 10.0.19045.4170]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Admin>cd C:\Users\Admin\Desktop\UNYT\DISTYS\DISTSYS 2024\SOFTWARE

C:\Users\Admin\Desktop\UNYT\DISTYS\DISTSYS 2024\SOFTWARE>java_ee_sdk-7-jdk7-windows-x64.exe -j "C:\Program Files\Java\jdk-1.8"

C:\Users\Admin\Desktop\UNYT\DISTYS\DISTSYS 2024\SOFTWARE>
```



Introduction

- Introduction**
- Installation Type
- Install Directory
- Update Tool
- Ready To Install
- Progress
- Config Results
- Summary

Welcome to the Java EE 7 SDK installation.

This installer will guide you through the installation process. You will shortly be able to learn the latest Java EE 7 features, and you can get started with the First Cup and Java EE Tutorials. View sample application source code and then deploy to GlassFish Server 4.0 to see them in action. You will find that Java EE 7 is a easy-to-learn feature-rich platform for developing web and enterprise applications.



ORACLE

Cancel

Back

Next



Installation Type

- Introduction
- Installation Type**
- Install Directory
- Update Tool
- Ready To Install
- Progress
- Config Results
- Summary



ORACLE

Choose installation type.

Typical Installation

Installs a GlassFish Server management domain; ideal for development or non business critical use. Please make sure that the ports 4848 and 8080 are free.

Custom Installation

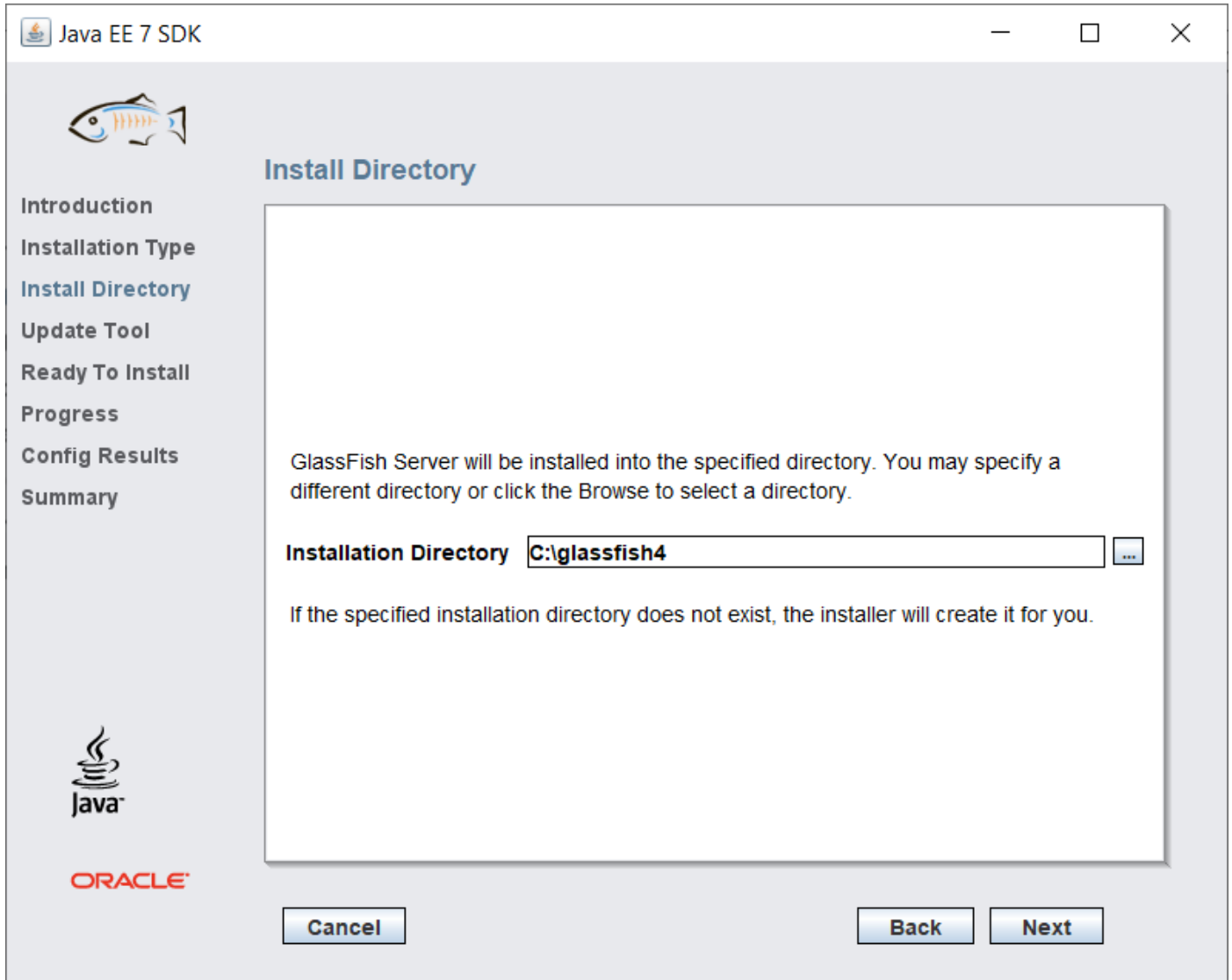
Not supported.

Cancel

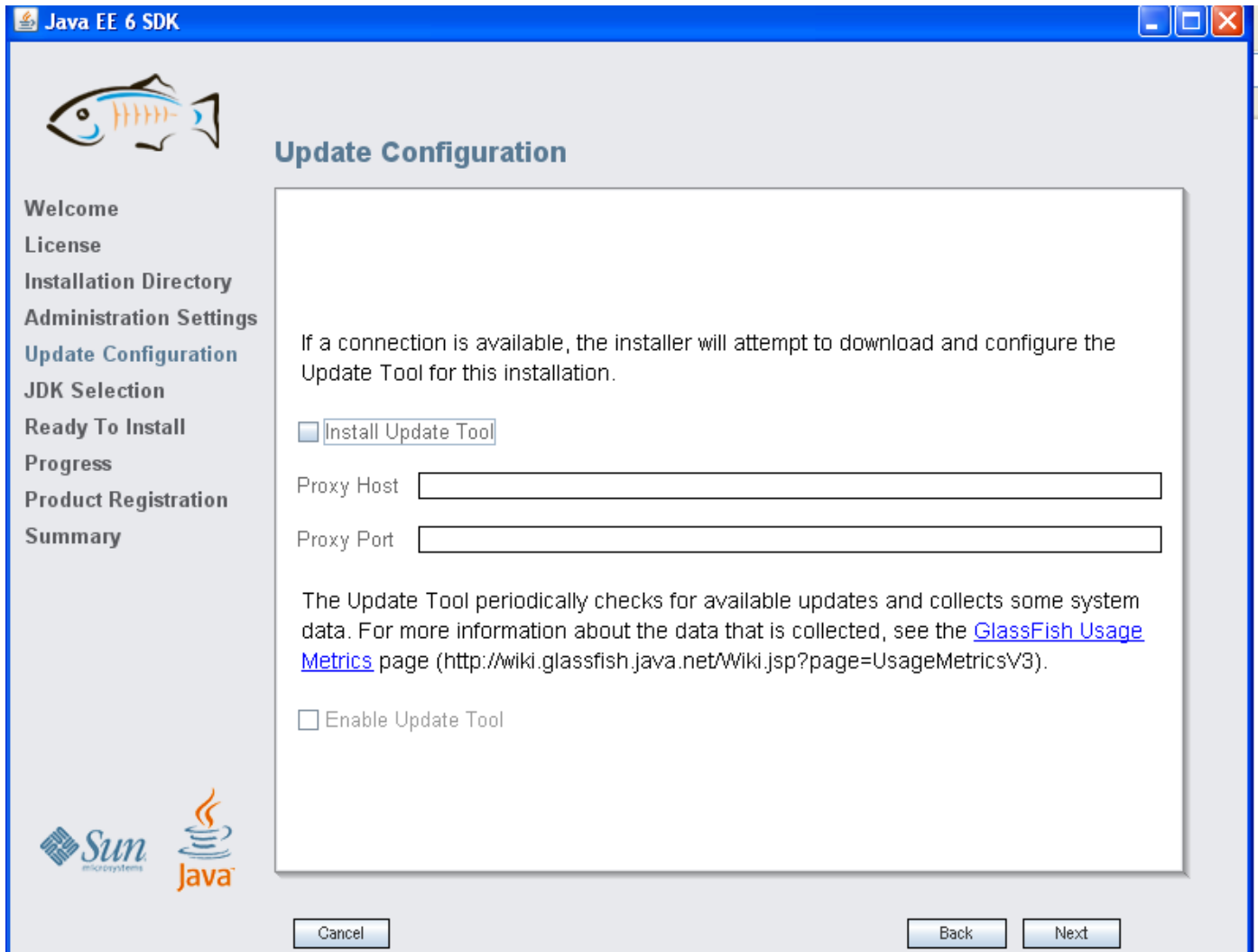
Back

Next

Choose directory for Glassfish:



Click again Next:





Ready To Install

- Introduction
- Installation Type
- Install Directory
- Update Tool
- Ready To Install**
- Progress
- Config Results
- Summary

Java EE 7 SDK

- Install JDK
- Install Update Tool Bootstrap
- Install GlassFish Server
- Install Uninstallation Software
- Configure Update Tool Bootstrap
- Configure GlassFish Server



ORACLE

Cancel

Back

Install



Progress

- Introduction
- Installation Type
- Install Directory
- Update Tool
- Ready To Install
- Progress**
- Config Results
- Summary

Java EE 7 SDK

Modular, Lightweight, Open

- Modular architecture based on OSGi
- Fast startup, less memory consumption
- Java EE 7 Certified
- Developed in Open Source



ORACLE

Installing GlassFish Server



Cancel

Back

Next



Config Results

- Introduction
- Installation Type
- Install Directory
- Update Tool
- Ready To Install
- Progress
- Config Results**
- Summary



ORACLE

The configuration has succeeded. Please see the output below.

**Domain domain1 allows admin login as user "admin" with no password.
Login information relevant to admin user name [admin]
for this domain [domain1] stored at
[C:\Users\Admin\gfclient\pass] successfully.
Make sure that this file remains protected.
Information stored in this file will be used by
administration commands to manage this domain.
Command create-domain executed successfully.**

Starting domain

Executing command :C:\glassfish4\glassfish\bin\asadmin.bat start-domain domain1

C:\glassfish4\glassfish\bin\asadmin.bat start-domain domain1
Attempting to start domain1.... Please look at the server log for more details.....

Cancel

Configure again

Next



Summary

- Introduction
- Installation Type
- Install Directory
- Update Tool
- Ready To Install
- Progress
- Config Results
- Summary



ORACLE

Overall Status: Complete

Please see the [detailed summary report](#) for an overview of this session, including [next steps](#) for using this installation. Please see the [log file](#) for detailed information.

[2024-04-11-14-47-install-summary.html](#)

[2024-04-11-14-47-install.log](#)

Product Name	Status
	Installed
Update Tool Bootstrap	Installed
GlassFish Server	Installed
Uninstallation Software	Installed
Update Tool Bootstrap	Configured
GlassFish Server	Configured

Cancel

Back

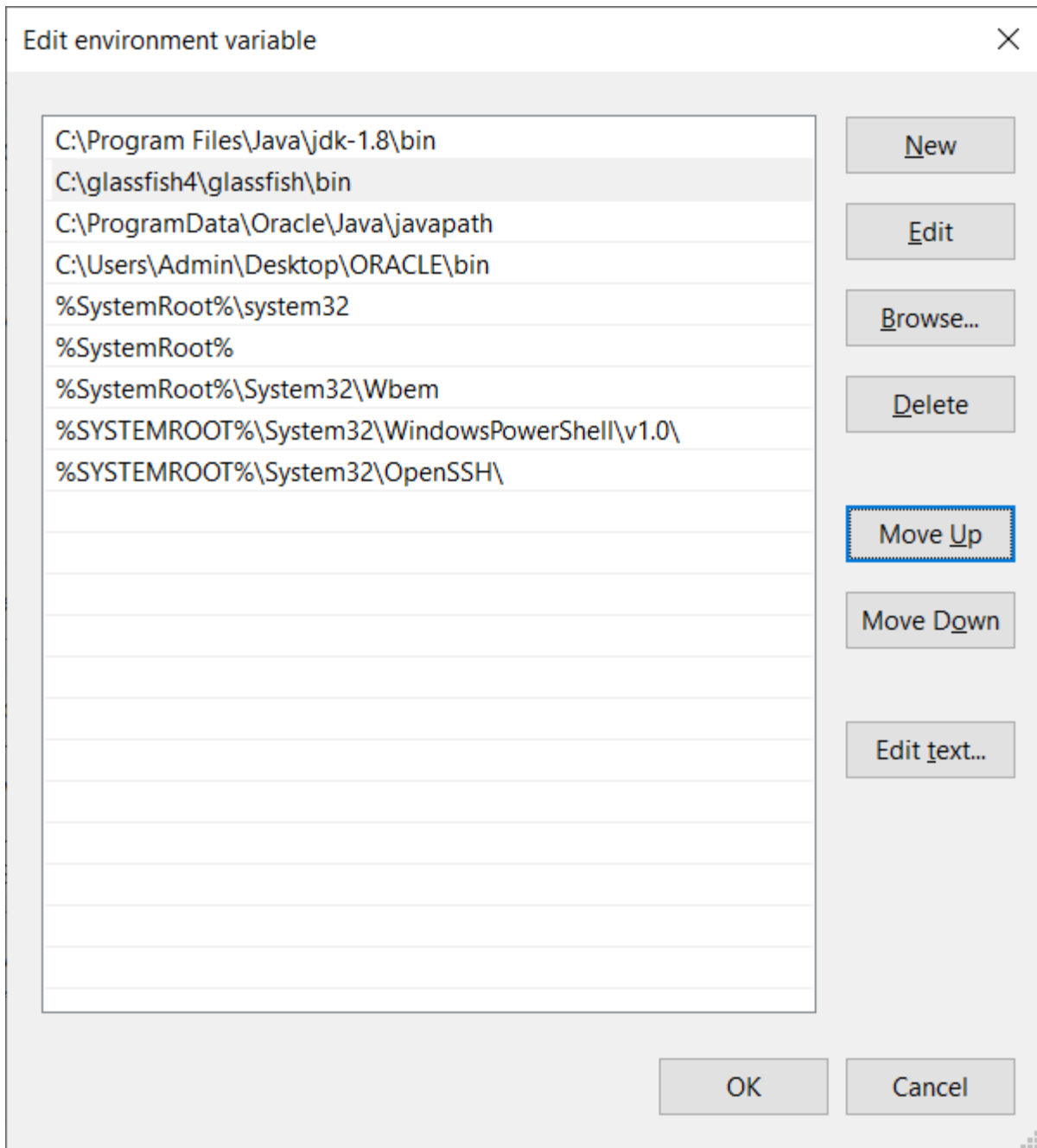
Exit

Only if “Overall Status” is “Complete”, your installation has been performed appropriately.

In order for the examples of this tutorial to execute you need to set the PATH with the directory of Glassfish as follows:

Click on Environment Variables:

Find the Path variable and click Edit.



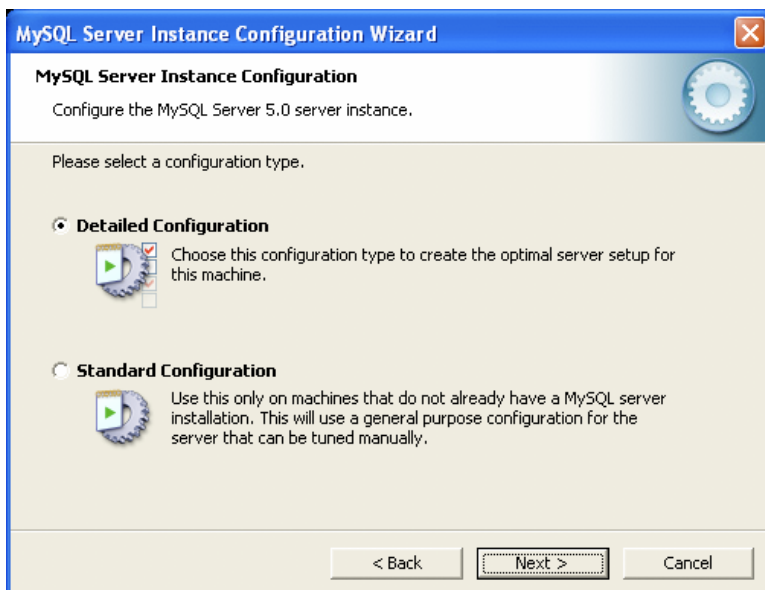
In the variable value add the path of Glassfish.

Download and Install MySQL Server:

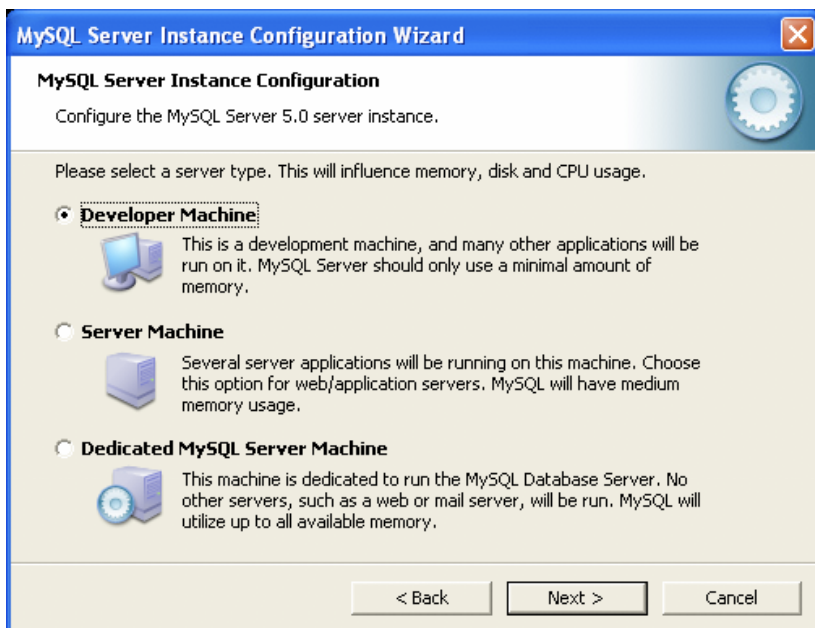
After you download use MySQL Server Instance Config Wizard



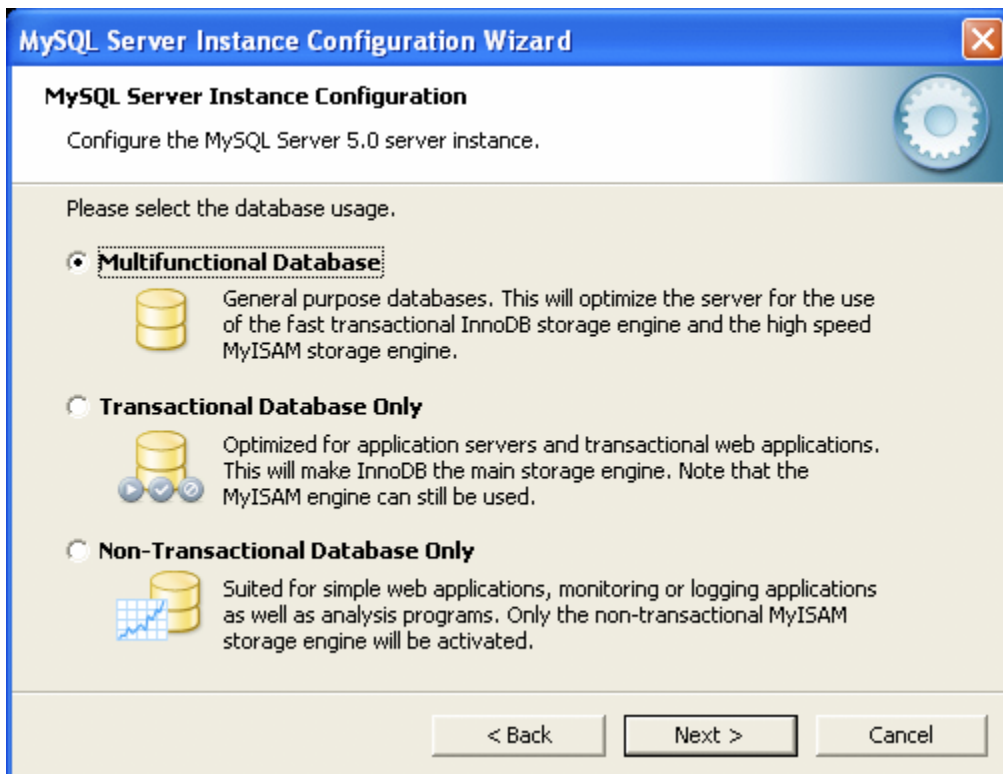
Choose the detailed configuration:



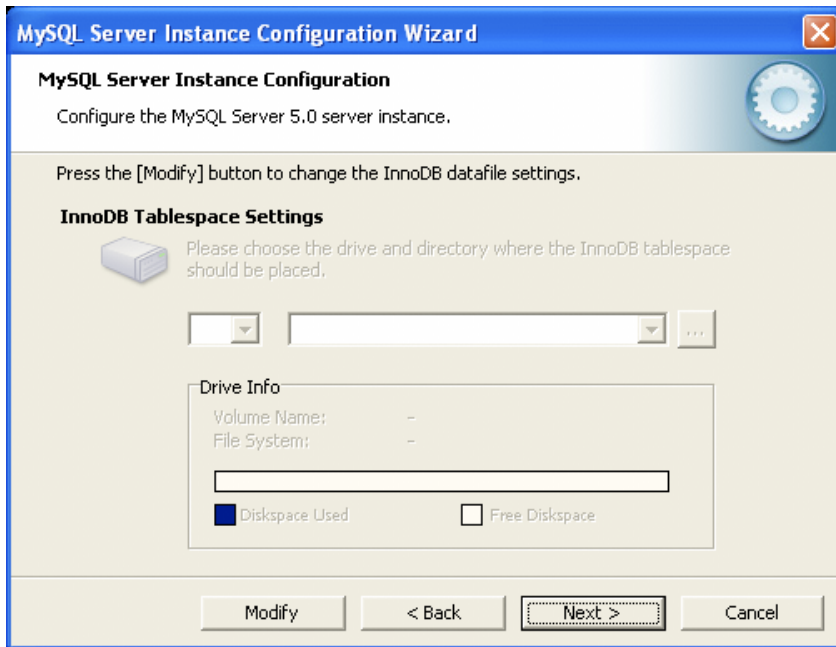
Choose developer machine:



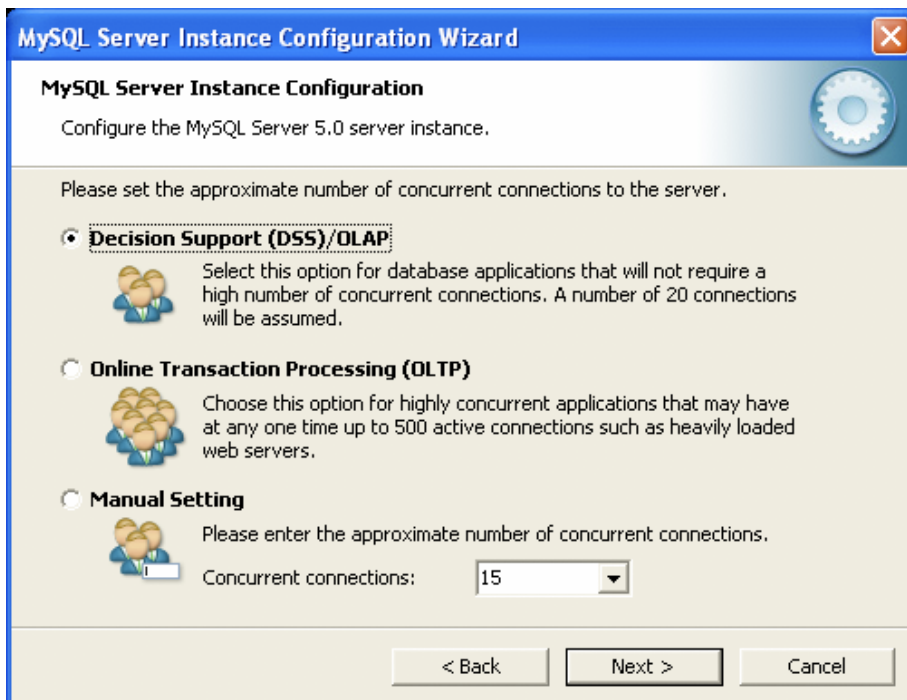
Choose multifunctional:



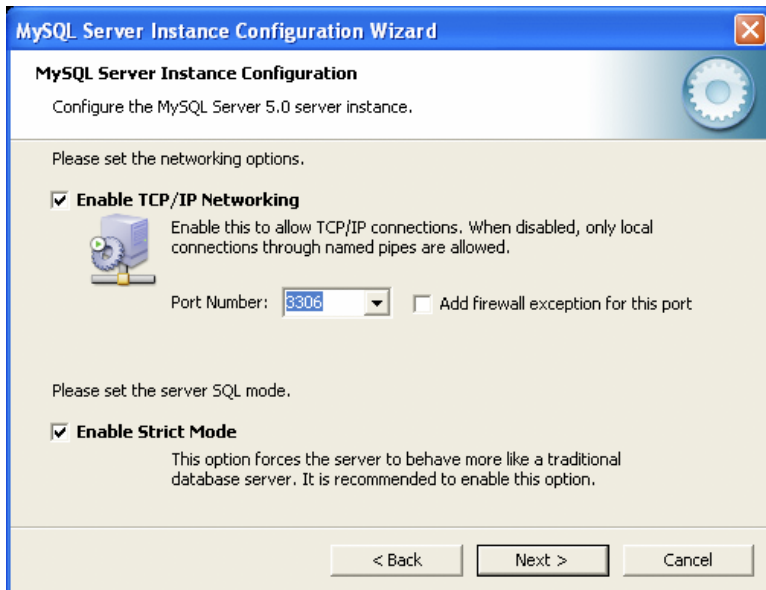
Choose Drive:



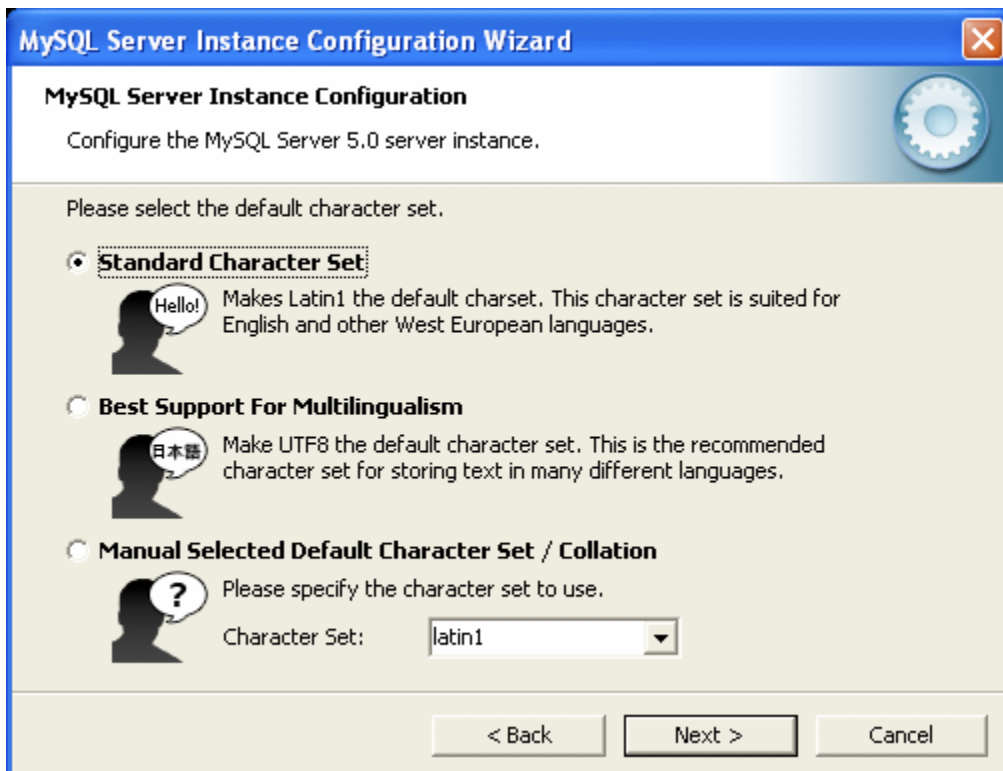
Choose decision support:



Perform the following checks:



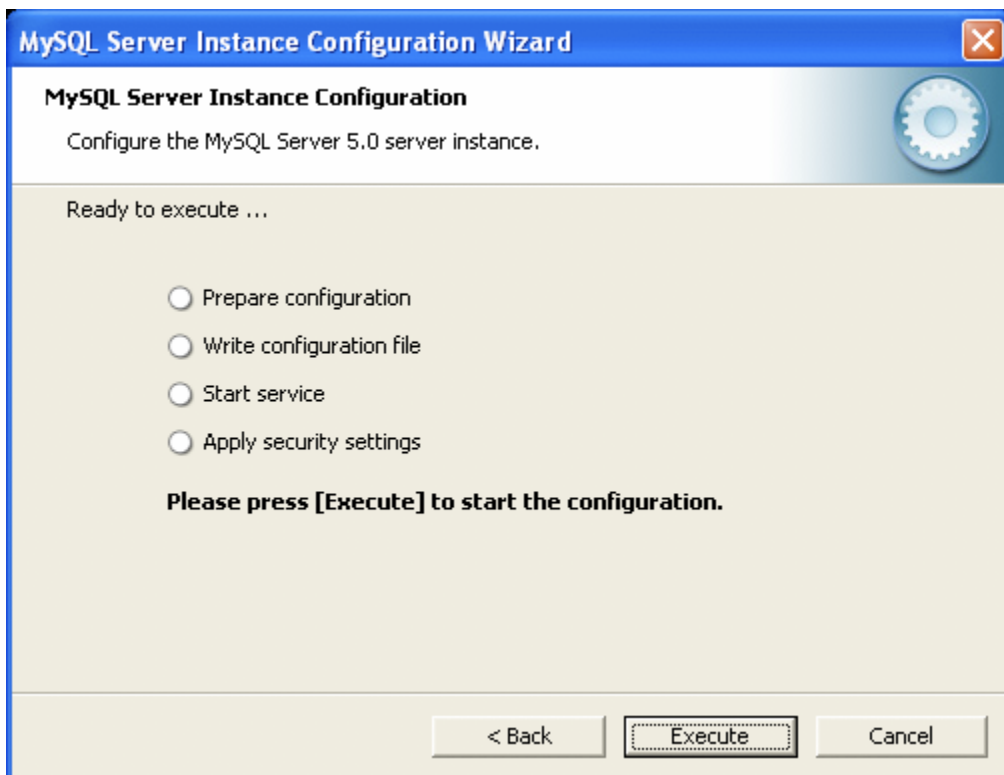
Best Support For Multilingualism: Choose this option if you want to use `utf8` as the default server character set. This is a Unicode character set that can store characters from many different languages.



Set the password for root:



Press Execute:



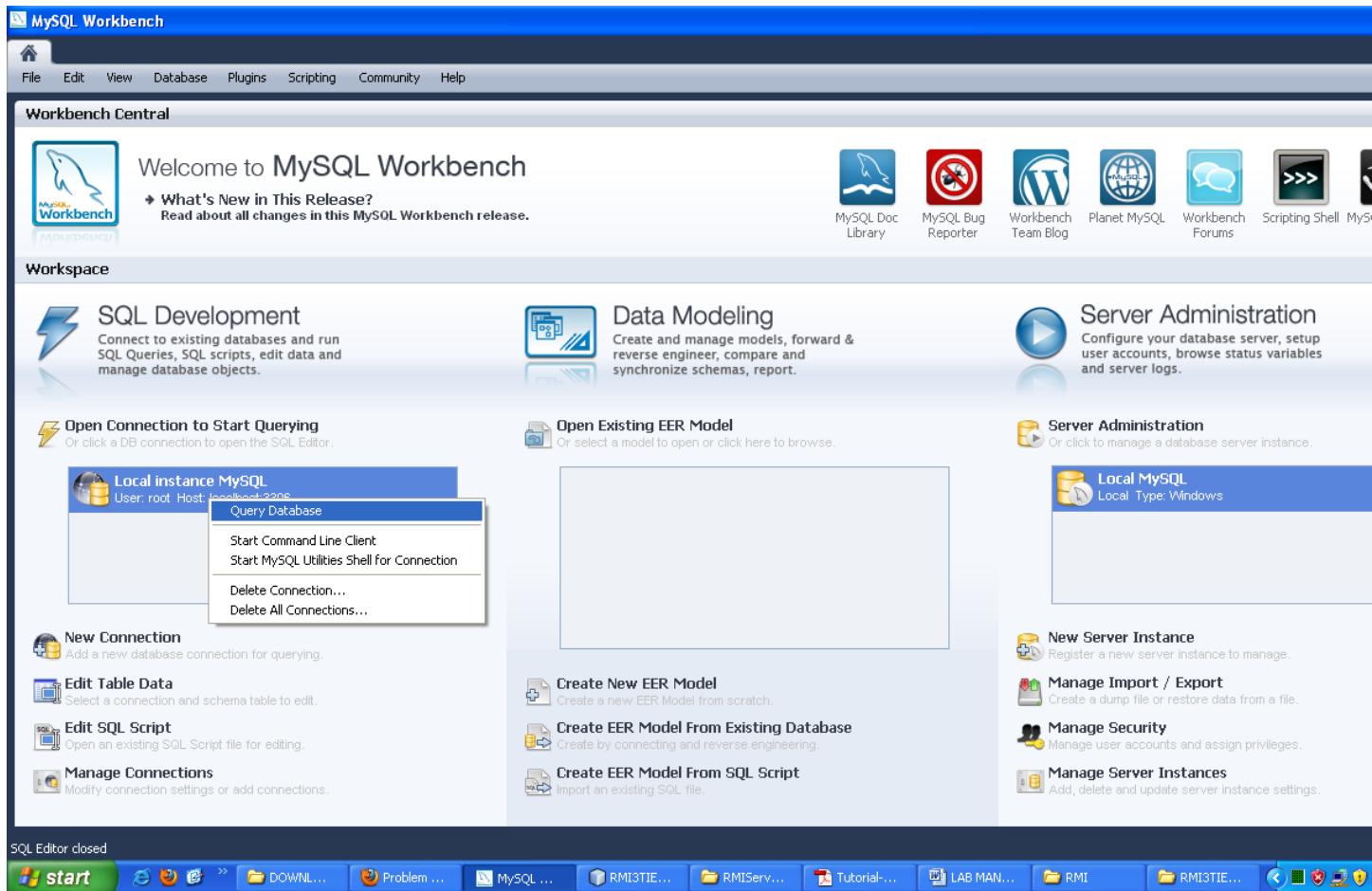
Restart the computer and the installation should be complete.

Install the MySQL Workbench.

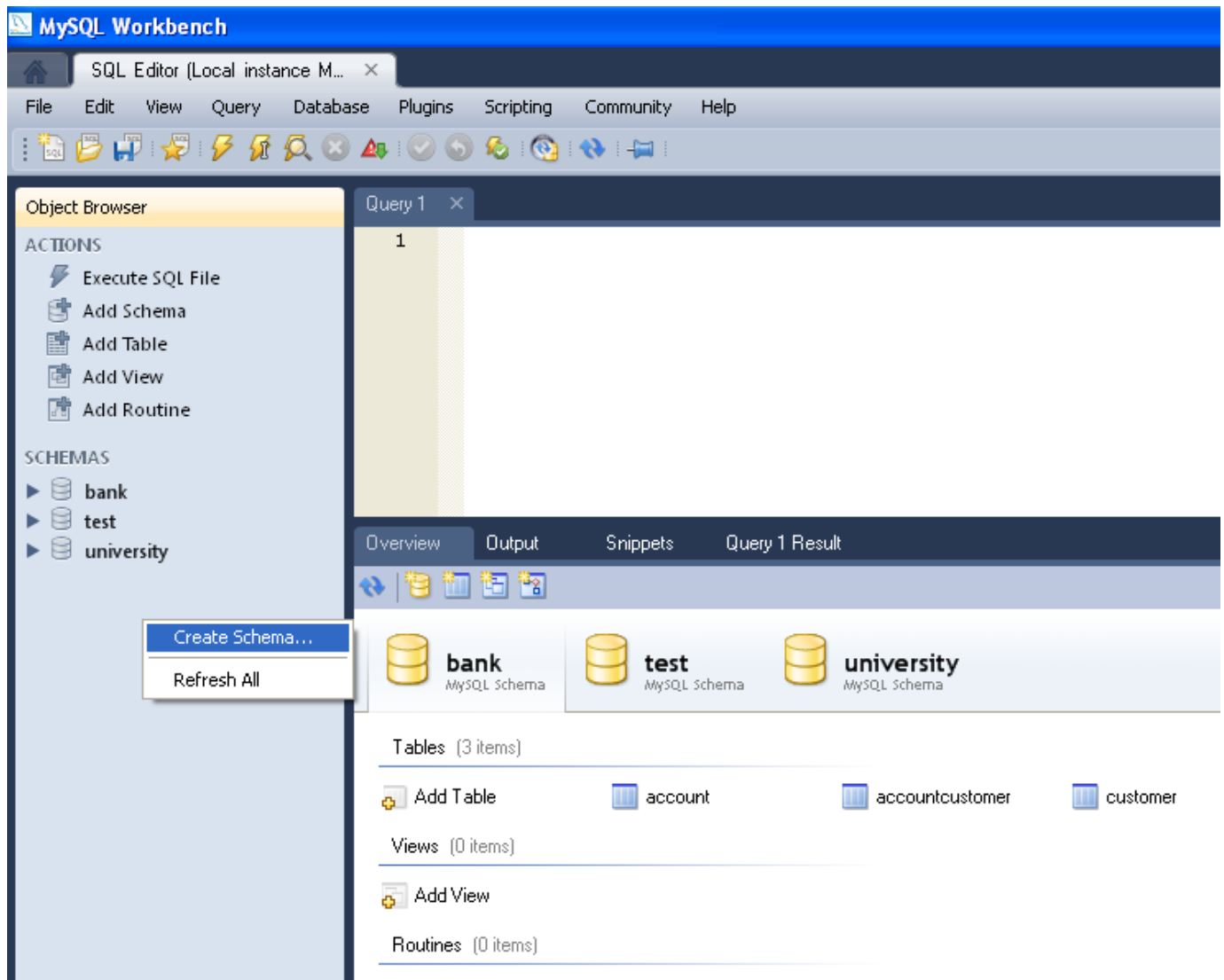
You can create the database in two ways:

1. By commands in the MySQL console
2. By graphical user interface in MySQL Workbench

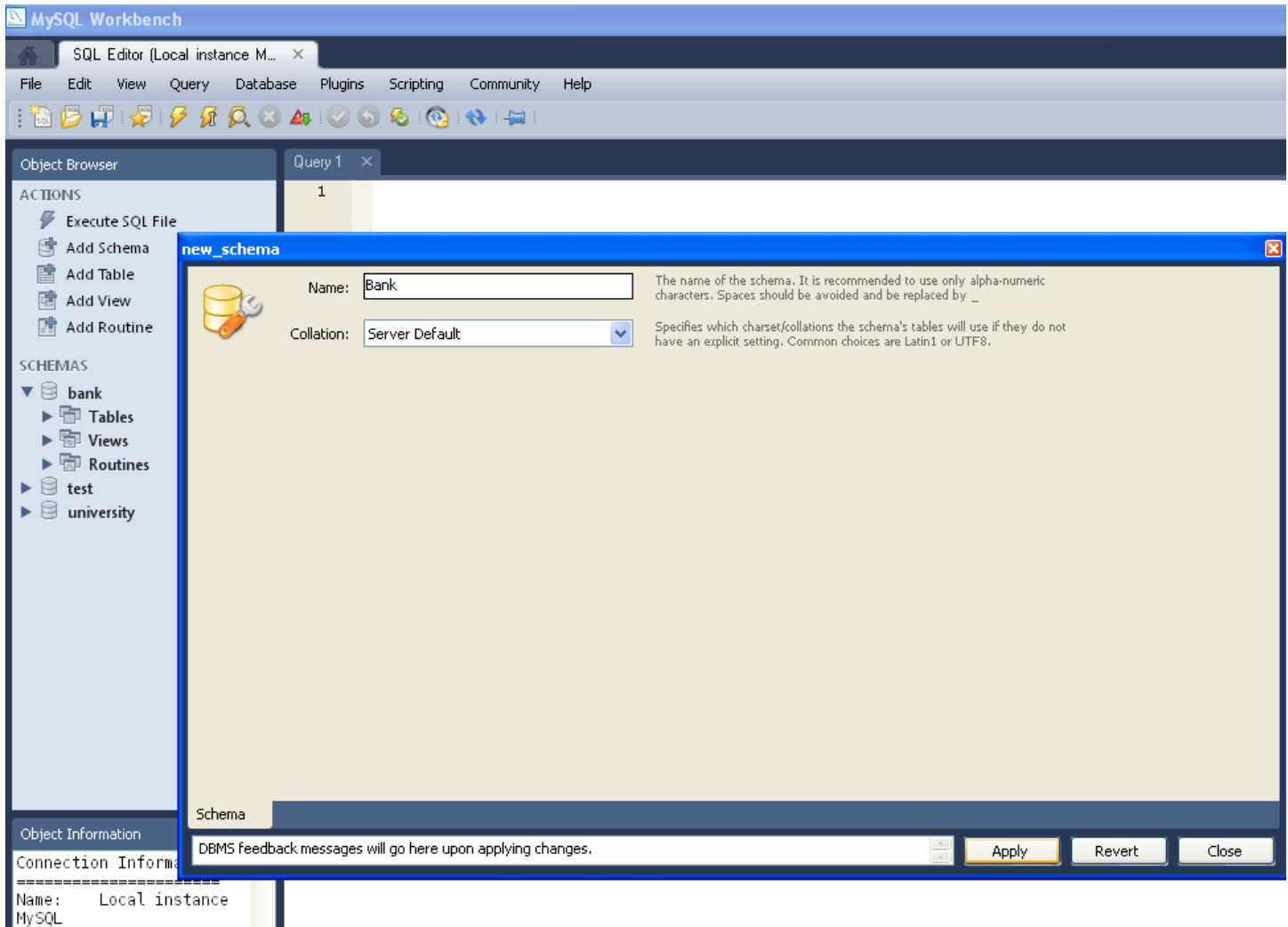
Click on local instance with the right and click Query Database.



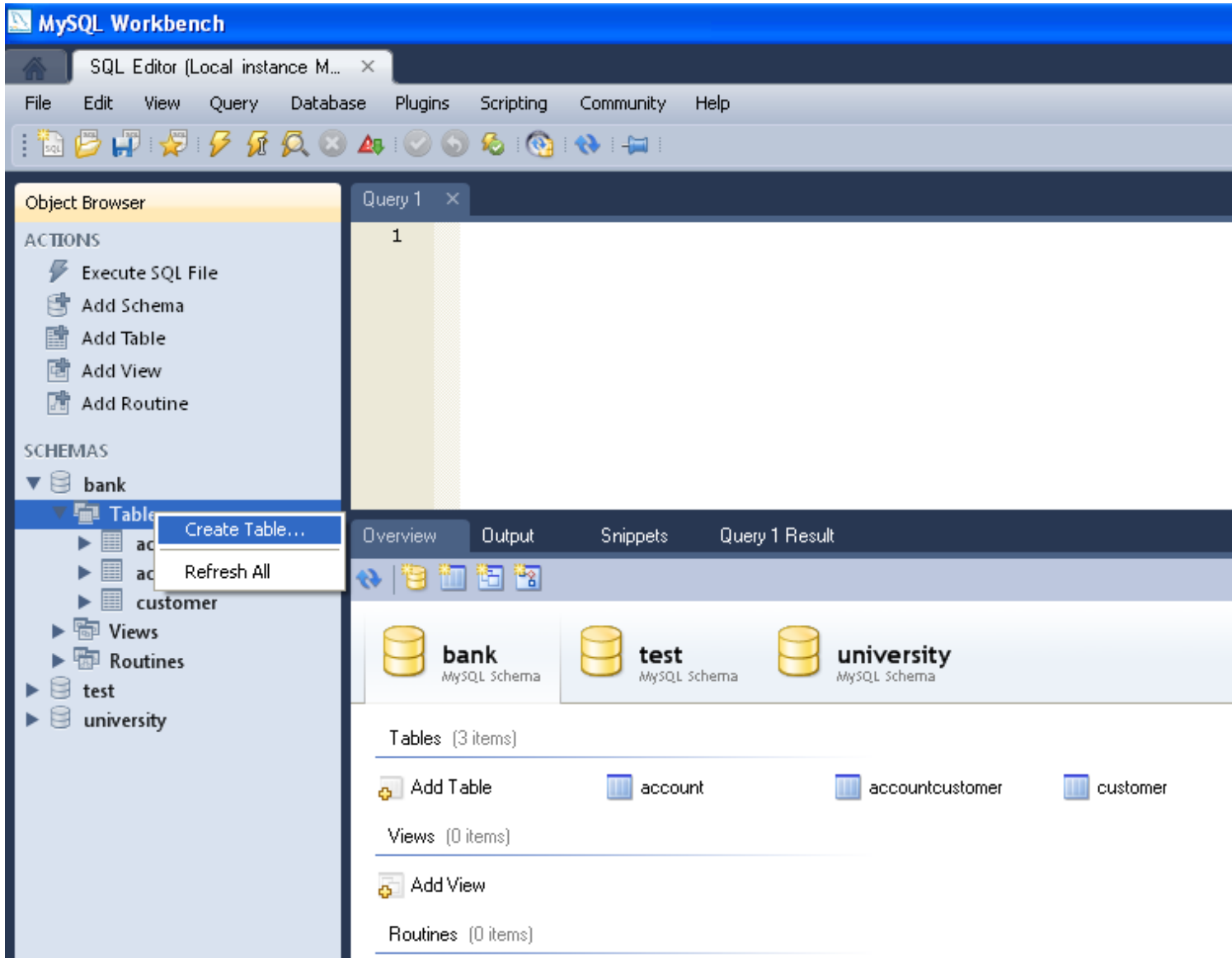
Click with the right and select create schema.



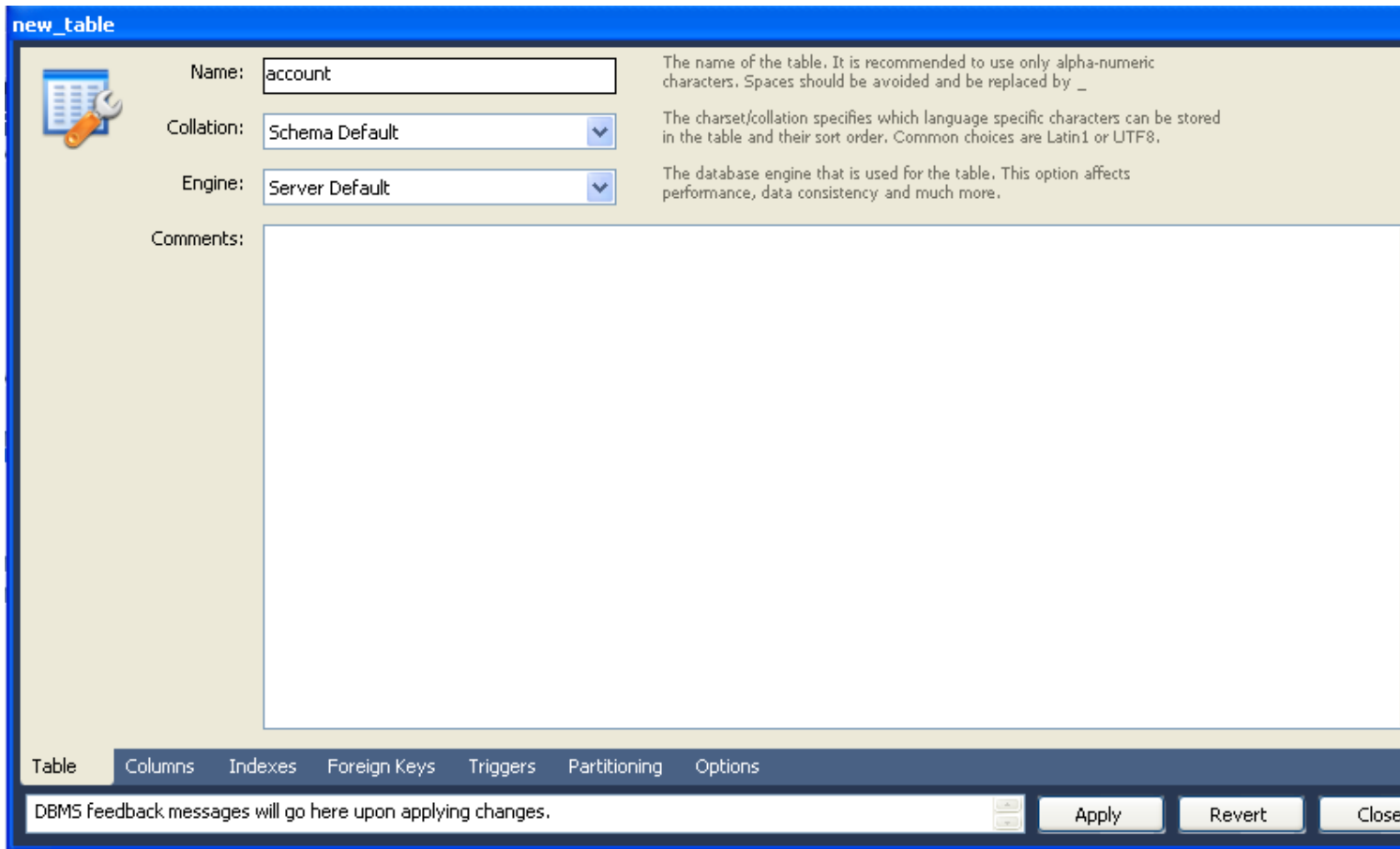
Give a name to the database: and press Apply.



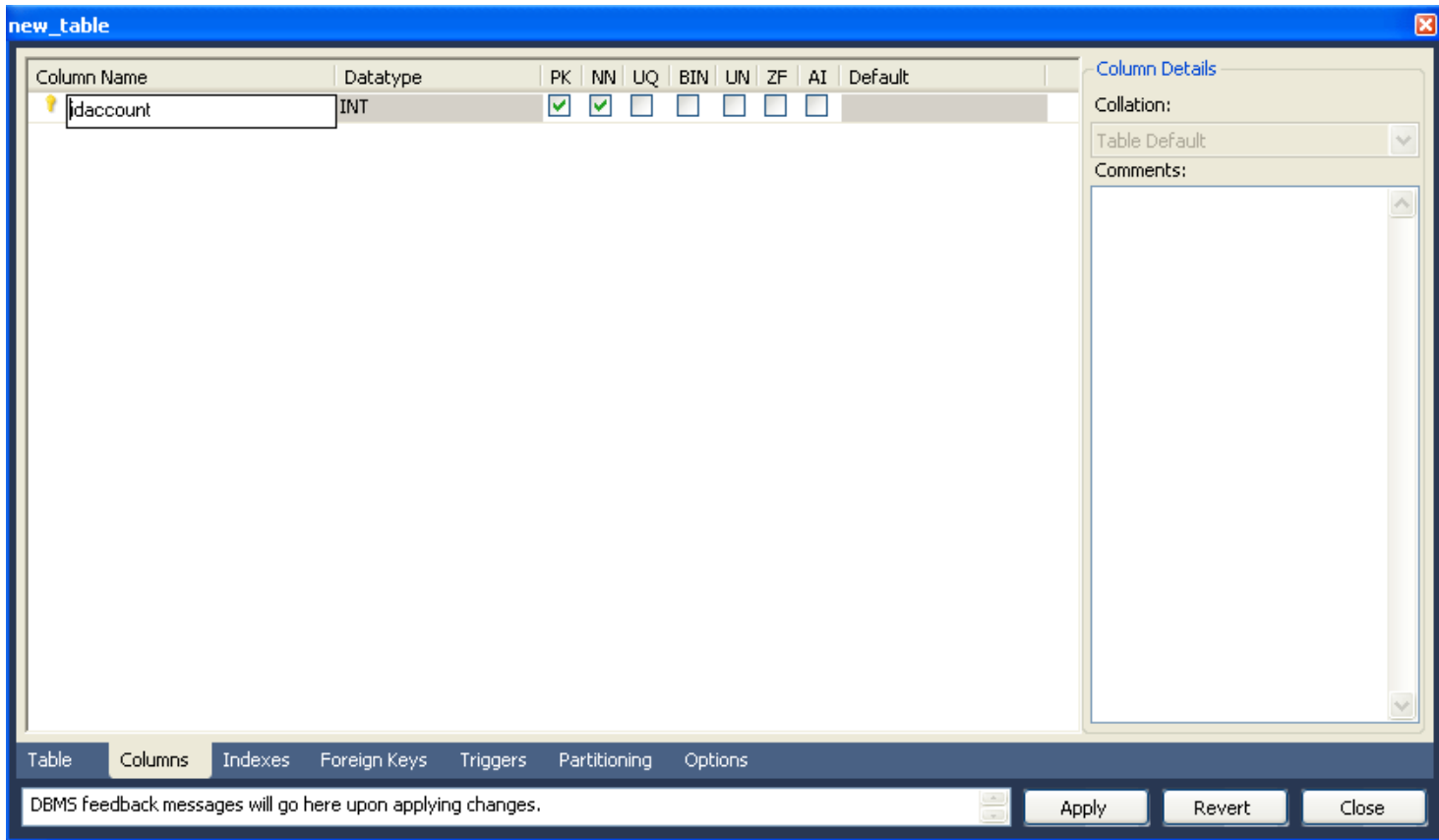
Click with the right on the Tables options and select Create Table:



Choose a name for the Table:



Click on Columns and add the columns for the table. At the end click Apply.



Following the above procedure create three tables:

Table Account

Fields: IdAccount (int), Balance (float)

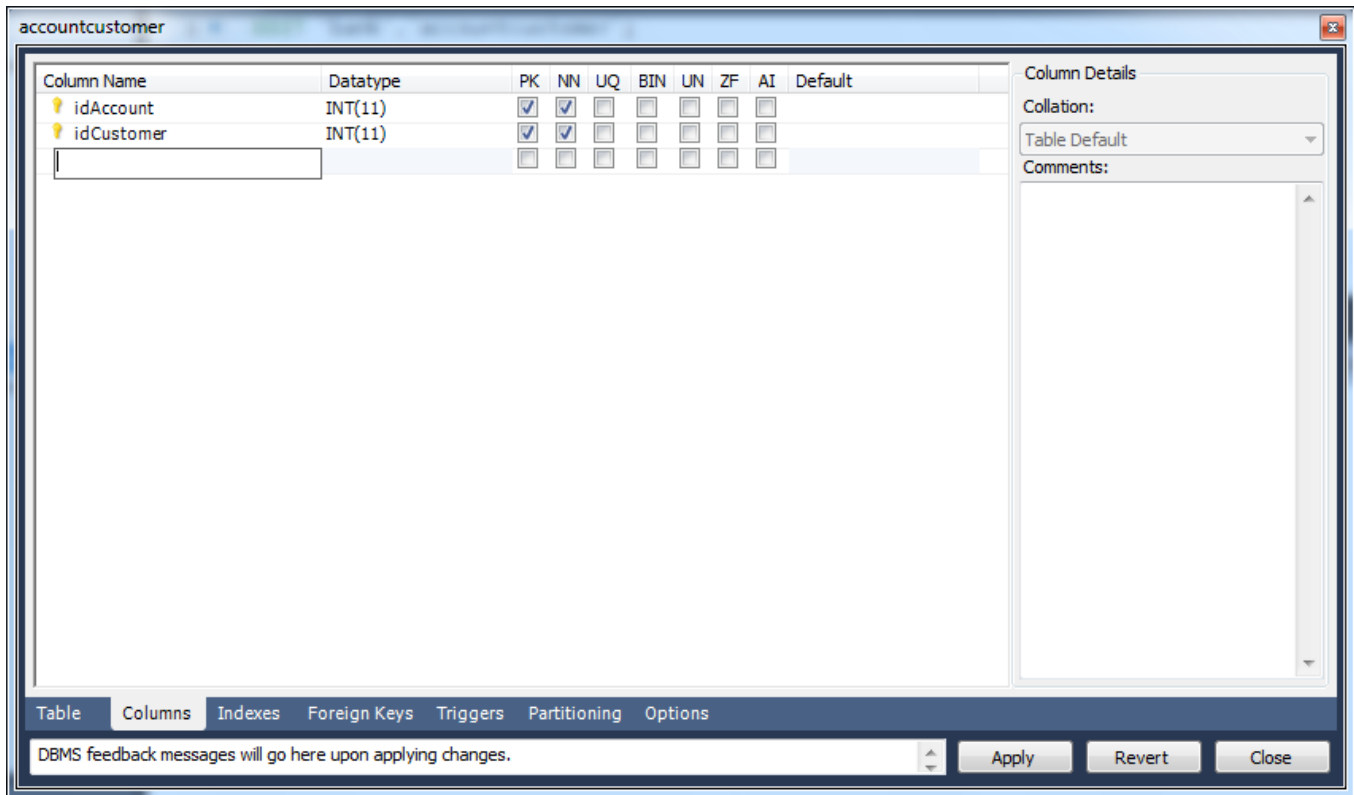
Table Customer

Fields: idCustomer(int), Name (Varchar), surname (Varchar)

Table AccountCustomer

Fields: idAccount, IdCustomer

In AccountCustomer both fields are primary keys as follows:

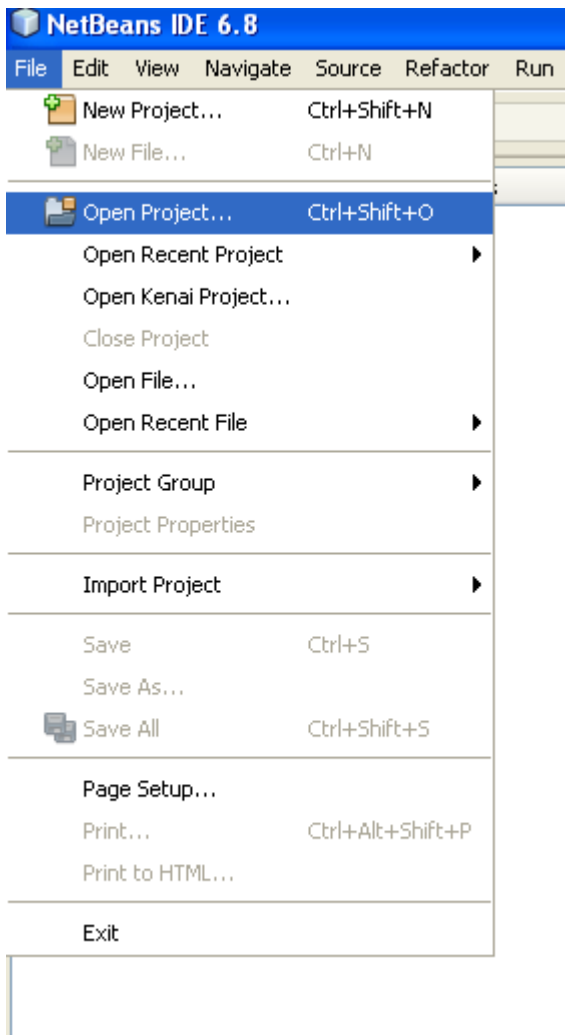


2. Running the Web examples

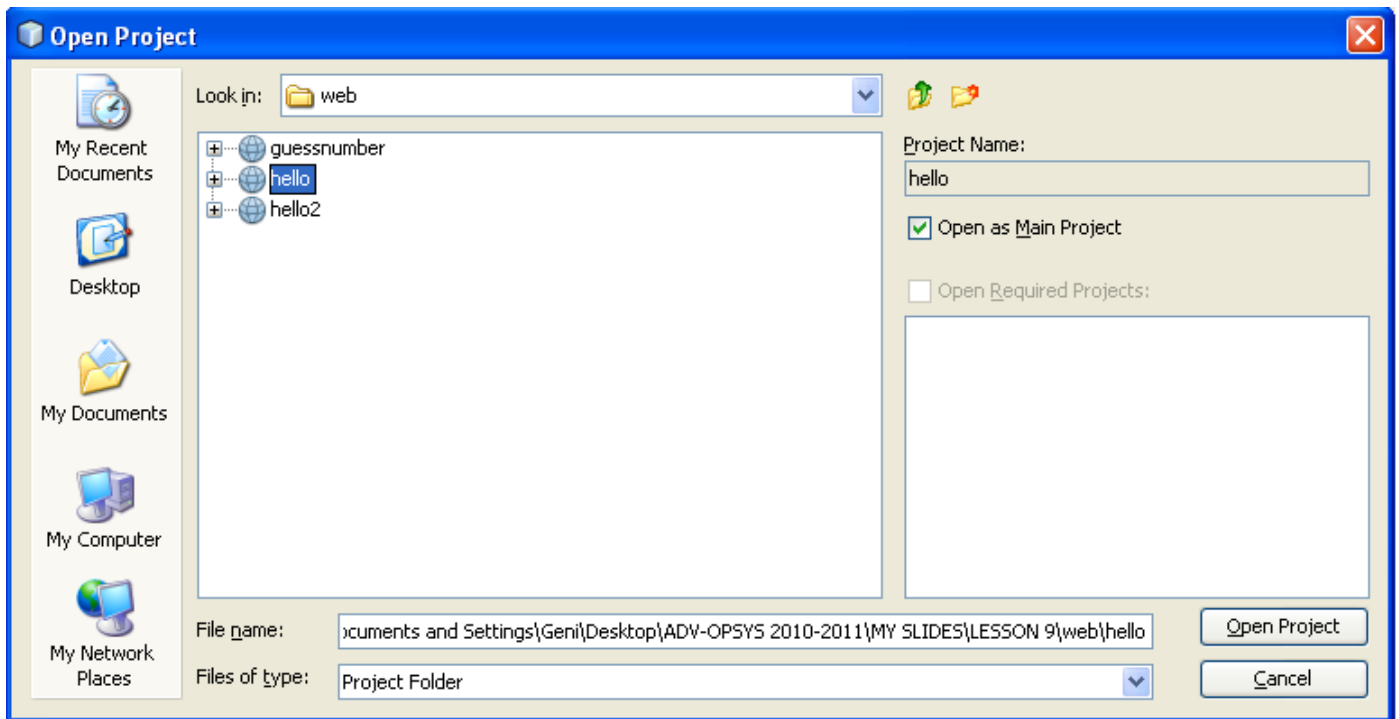
You will be given a folder with the following examples:

- Hello: a web application that shows with a facelet a greeting that is read from a java bean
- Hello2: a web application that reads a string from the user in the browser through a servlet, sends it to another servlet that returns back a response
- GuessNumber: a web application that reads a number through a facelet and compares the number with a random number generated with a java bean. Then it sends back the response to another facelet.

To run the examples go to Open Project as follows:



Open the Web folder:

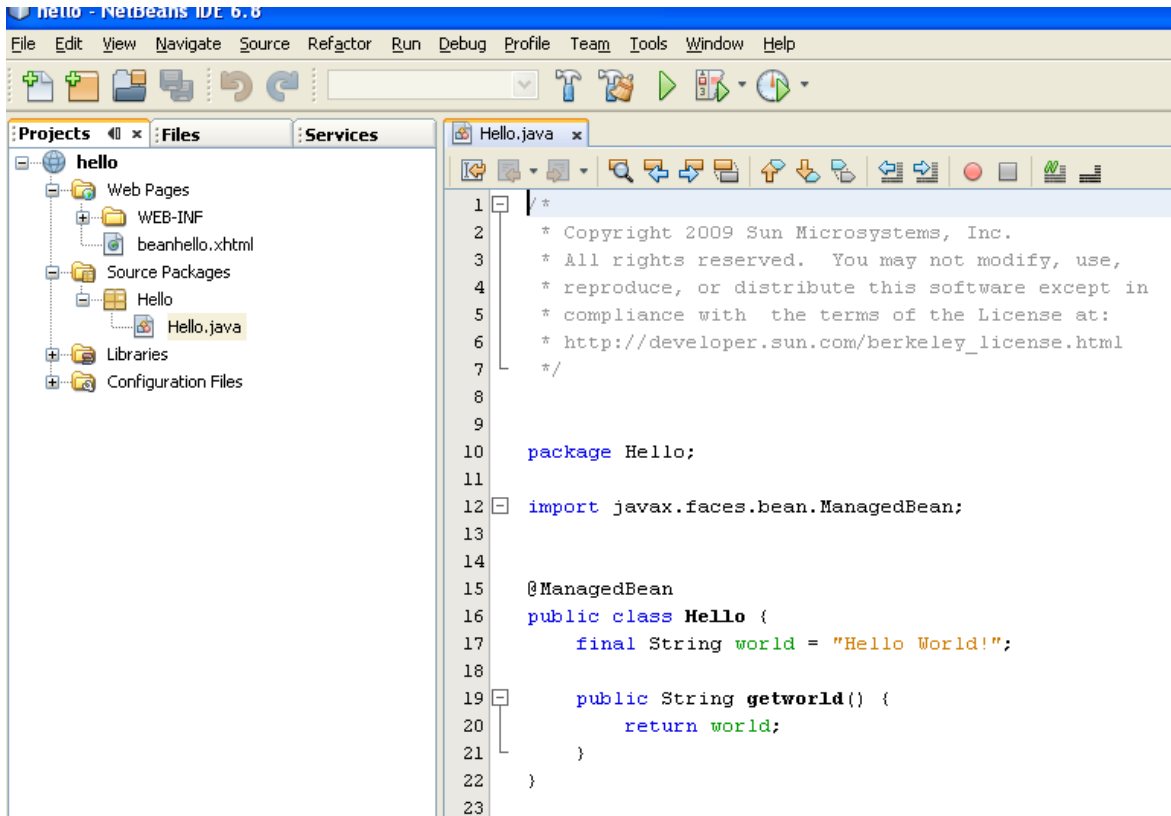


And choose the project that you want to open.

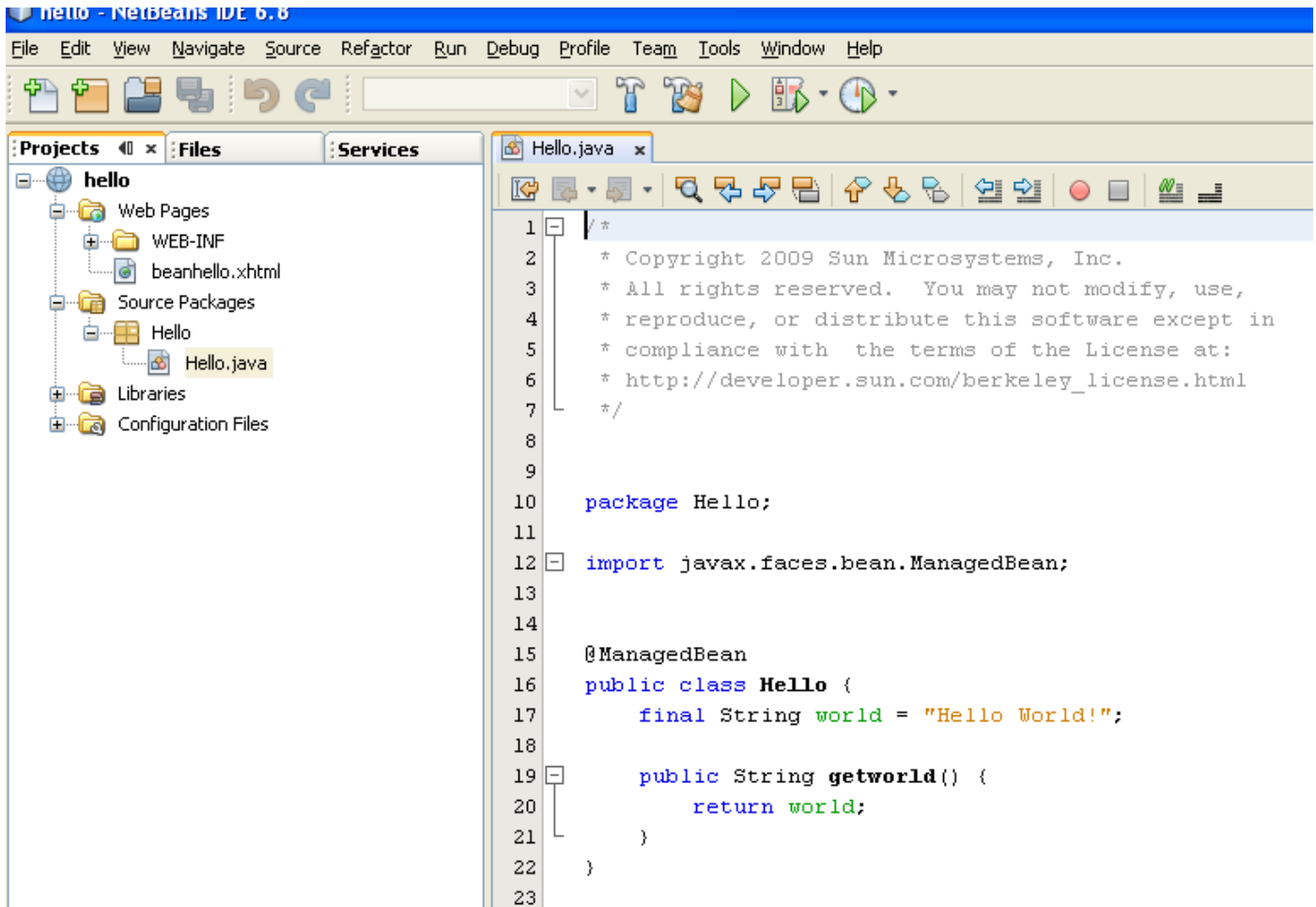
Example 1: Connecting a Bean with a Facelet

The project is composed as follows:

One Bean that prints the String “Hello World” as shown below:

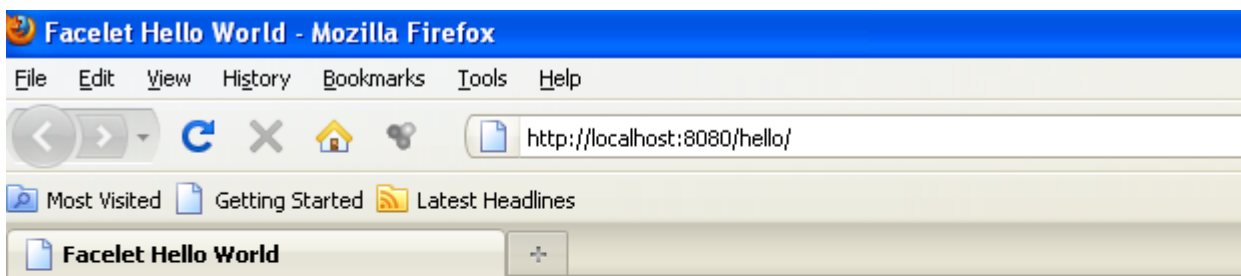


And one Facelet that takes the message of the bean and shows it in the web page.



If we want to run the application, it must be first build, then deployed and finally run. To properly build copy the given directory “bp-project” in the same directory where the “web” folder is.

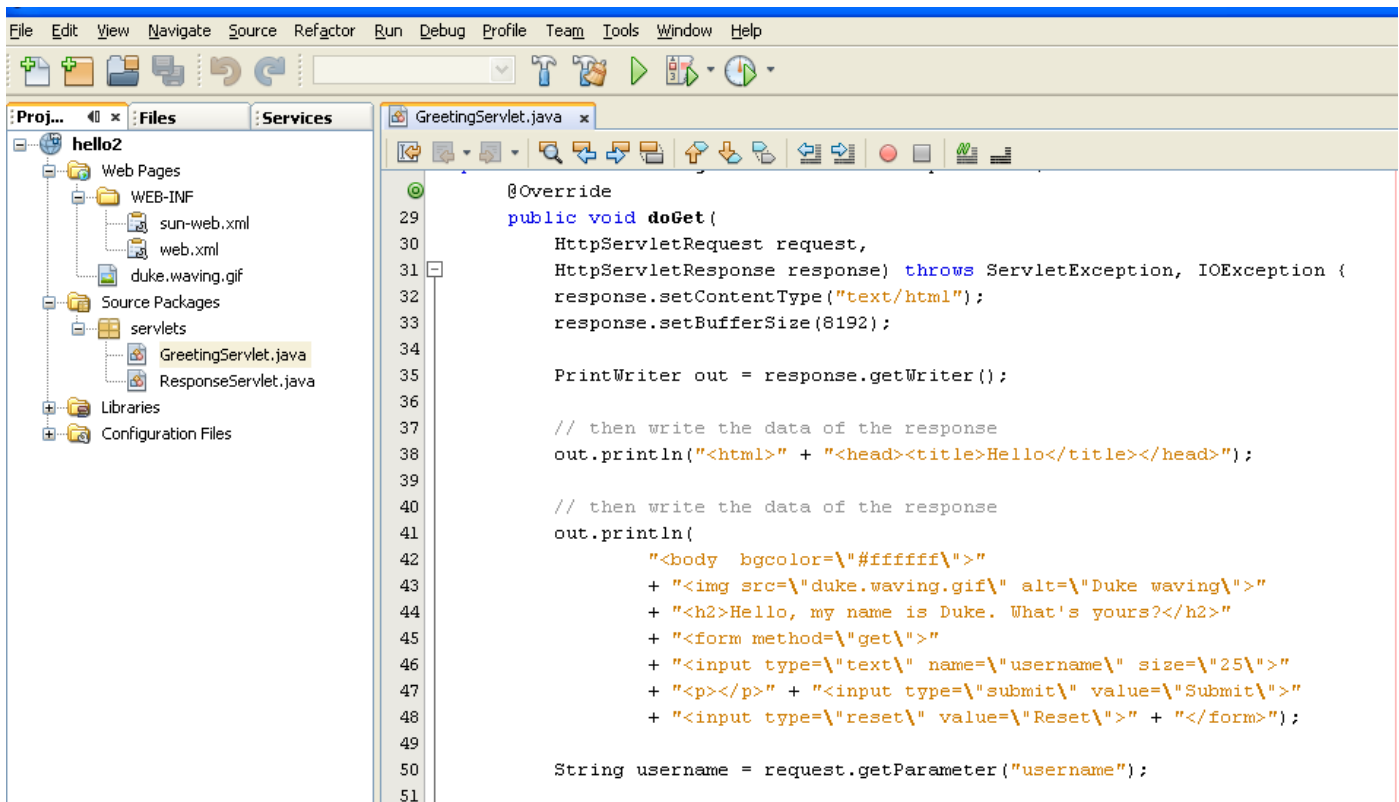
Then perform in turn: Clean and Build, Deploy and Run. When it is executed the following window will appear:



Hello World!

Example 2: Hello2

Once you open the web application you will see the following:

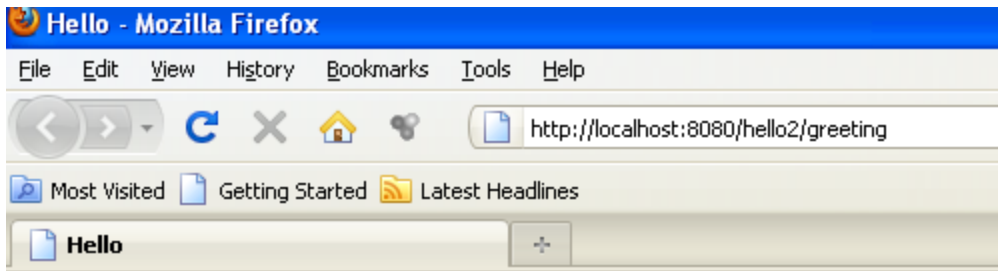


```
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
:Proj... x :Files :Services GreetingServlet.java x
hello2
  Web Pages
  WEB-INF
  sun-web.xml
  web.xml
  duke.waving.gif
  Source Packages
  servlets
  GreetingServlet.java
  ResponseServlet.java
  Libraries
  Configuration Files

@Override
29 public void doGet (
30     HttpServletRequest request,
31     HttpServletResponse response) throws ServletException, IOException {
32     response.setContentType("text/html");
33     response.setBufferSize(8192);
34
35     PrintWriter out = response.getWriter();
36
37     // then write the data of the response
38     out.println("<html>" + "<head><title>Hello</title></head>");
39
40     // then write the data of the response
41     out.println(
42         "<body bgcolor=\"#ffffff\">"
43         + "<img src=\"duke.waving.gif\" alt=\"Duke waving\">"
44         + "<h2>Hello, my name is Duke. What's yours?</h2>"
45         + "<form method=\"get\">"
46         + "<input type=\"text\" name=\"username\" size=\"25\">"
47         + "<p></p>" + "<input type=\"submit\" value=\"Submit\">"
48         + "<input type=\"reset\" value=\"Reset\">" + "</form>");
49
50     String username = request.getParameter("username");
51
```

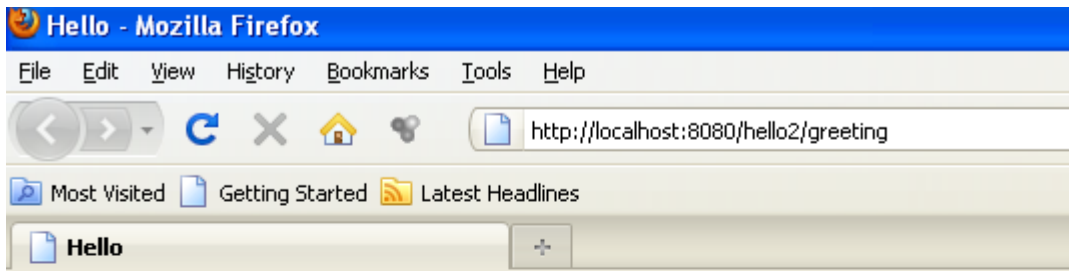
There are two servlets: one to get the input from the user GreetingServlet and one to send the response to the user ResponseServlet:

Once we build, deploy and run the application we will have the following:



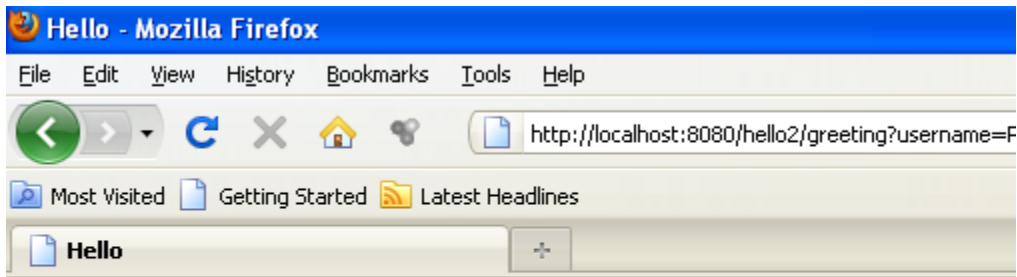
Hello, my name is Duke. What's yours?

If we write something in the textfield, the greeting servlet will read the data and send it to the response servlet that will send a response to the client as follows:



Hello, my name is Duke. What's yours?

If we press submit the following appears:



Hello, my name is Duke. What's yours?

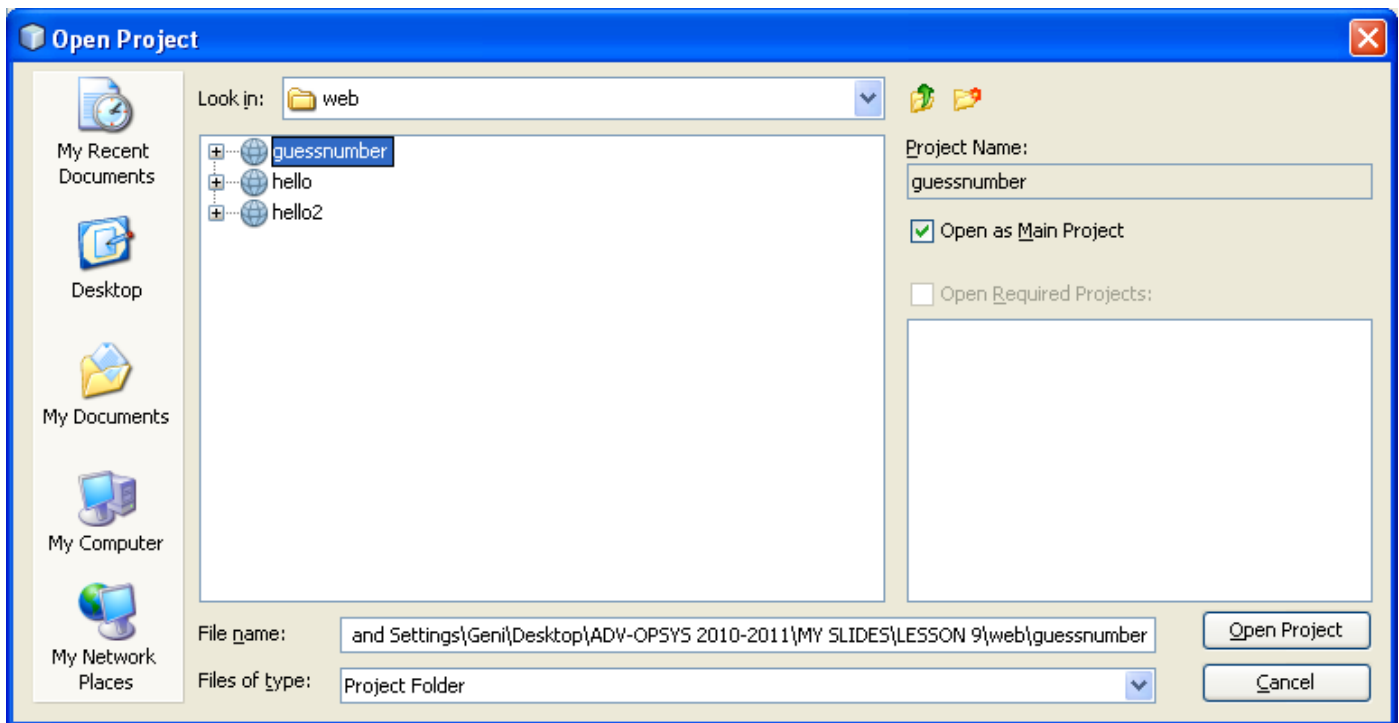
Submit

Reset

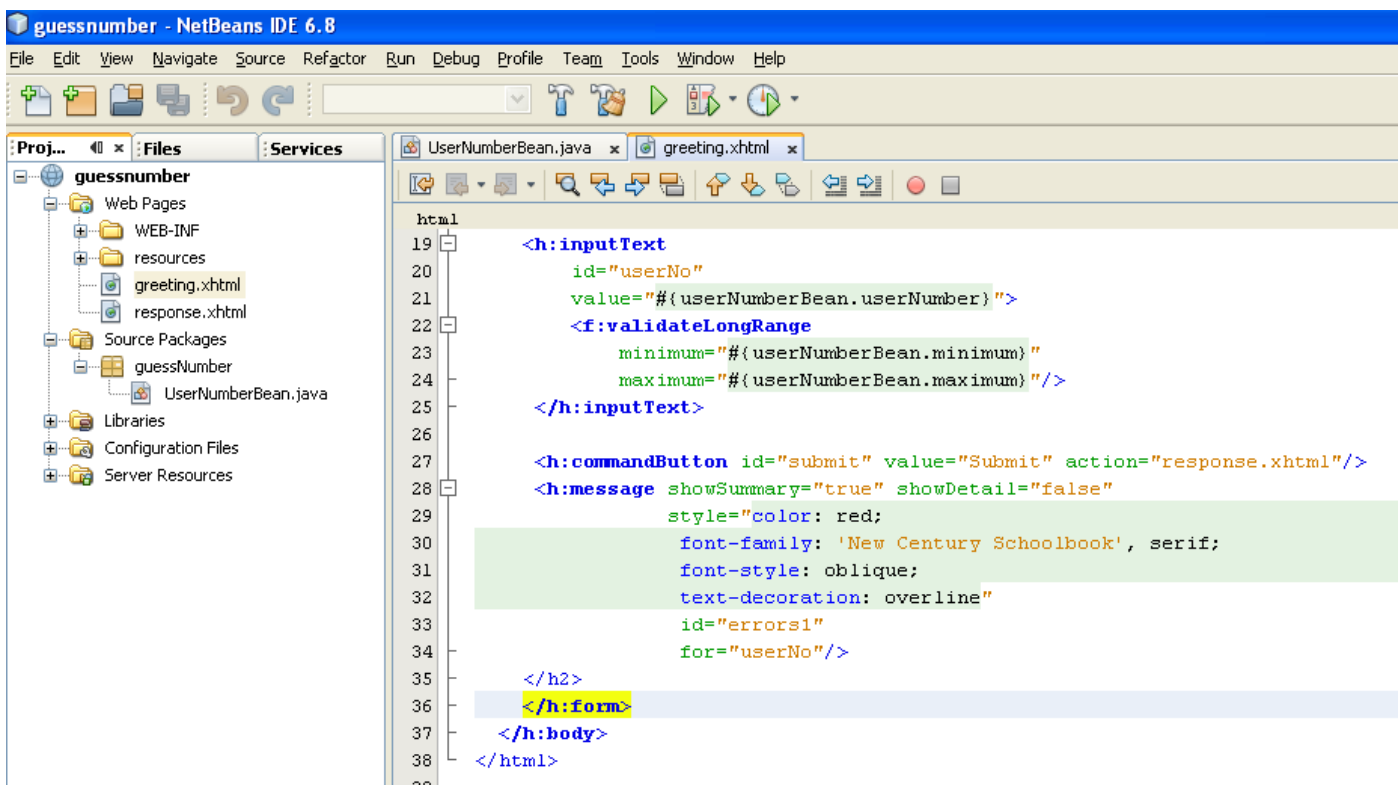
Hello, Peter Pan!

Example 3: Guess number

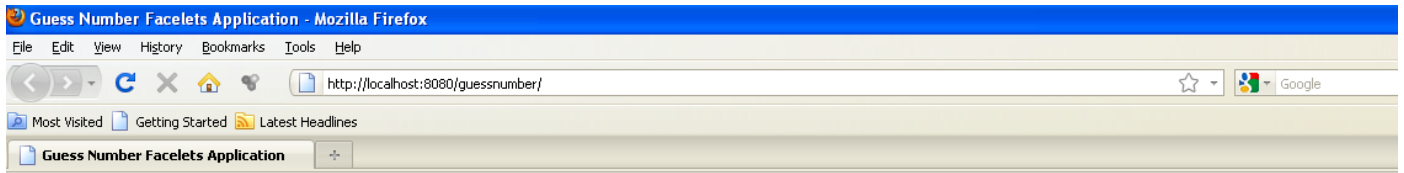
Open the project:



In the project there is one bean and two facelets: one to read the value and one to display the response from the bean.

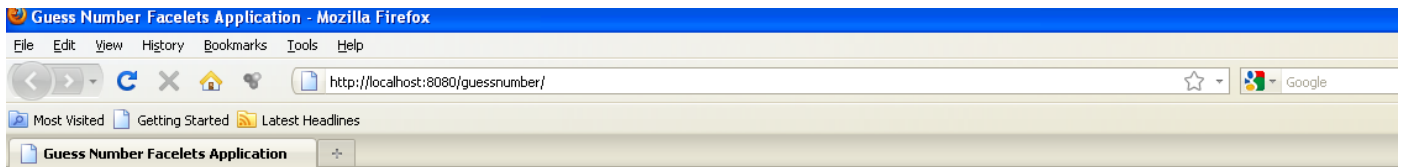


Once you build, deploy and run the application you will see the following:



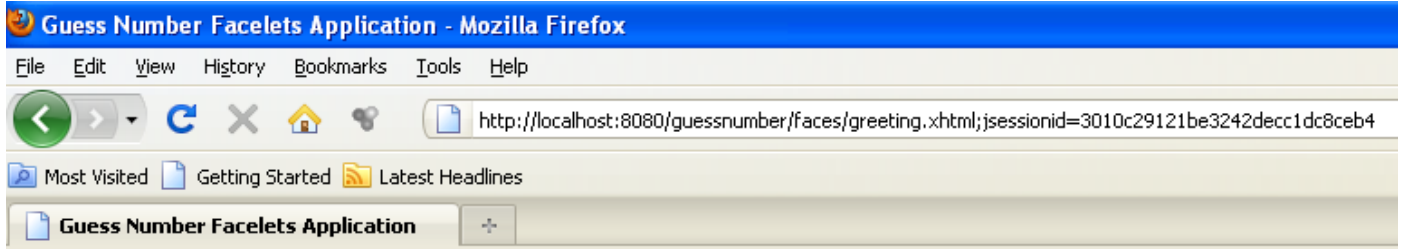
Hi, My name is Duke. I am thinking of a number between 0 and 10. Can you guess it ?

Insert a number:



Hi, My name is Duke. I am thinking of a number between 0 and 10. Can you guess it ?

Press submit:



Sorry, 7 is incorrect.

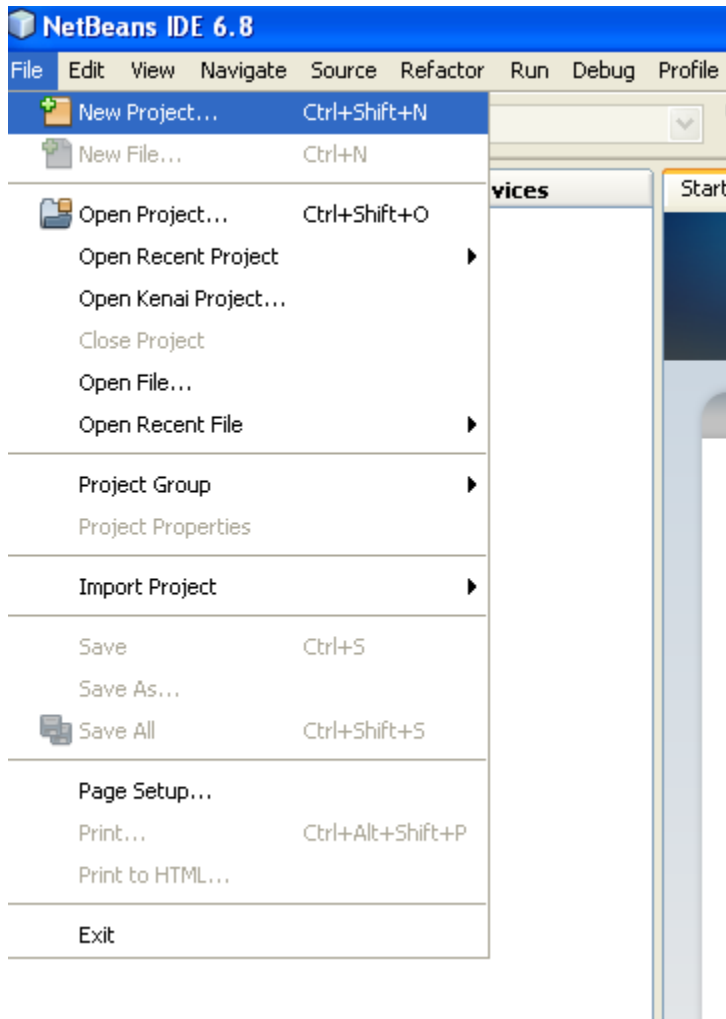
[Back](#)

The numbers do not match.

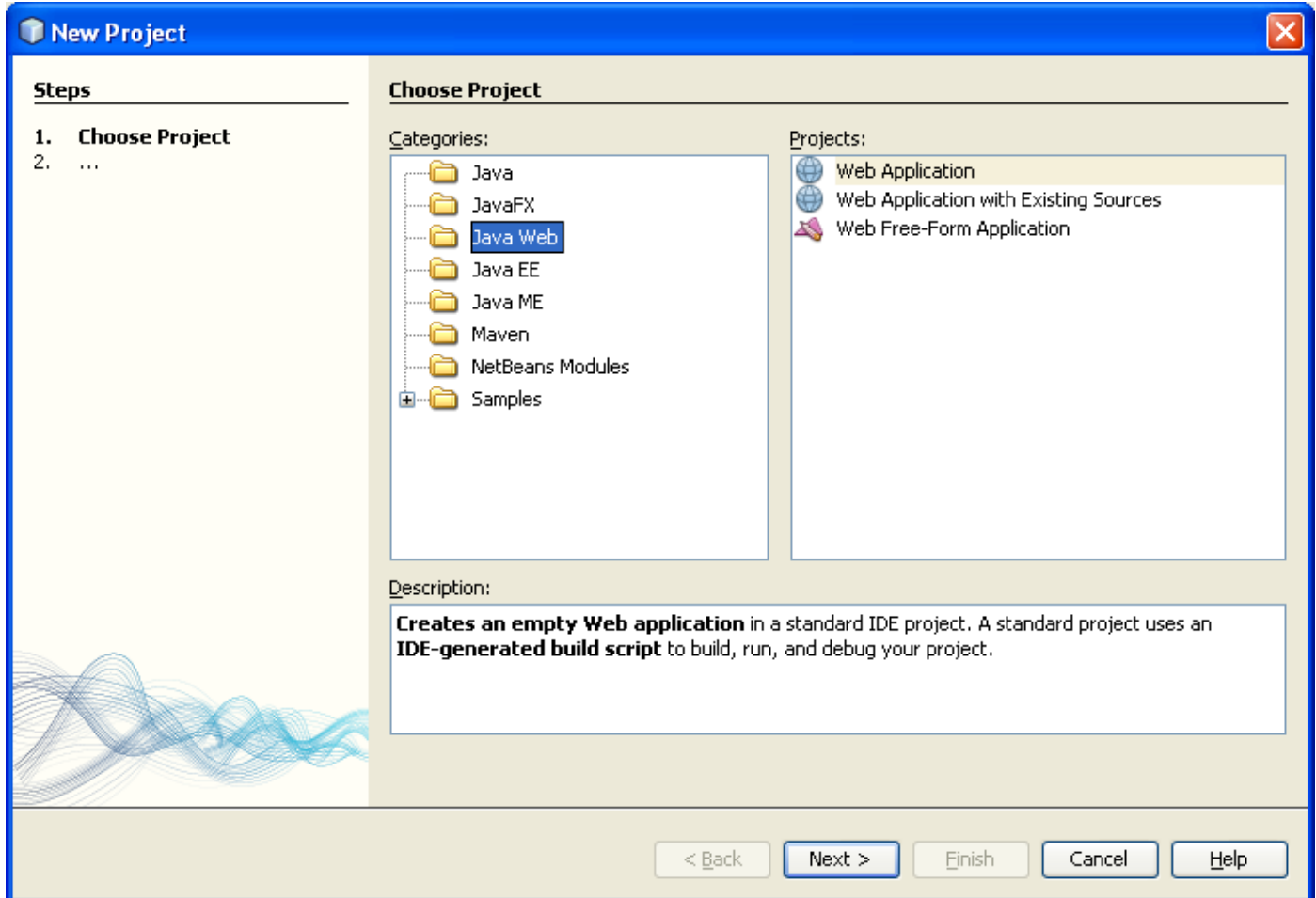
3. Developing a web application with a Servlet and an EJB

In order to create the web Application perform the following steps in NetBeans:

Go to new in Netbeans:



Select Java Web as follows:



New Web Application [Close]

Steps

1. Choose Project
- 2. Name and Location**
3. Server and Settings
4. Frameworks

Name and Location

Project Name:

Project Location:


Project Folder:

Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries
(see Help for details).

Set as Main Project



New Web Application



Steps

1. Choose Project
2. Name and Location
- 3. Server and Settings**
4. Frameworks

Server and Settings

Add to Enterprise Application: <None>

Server: GlassFish v3 Domain

Use dedicated library folder for server JAR files

Java EE Version: Java EE 6 Web

Context Path: /HelloWorld

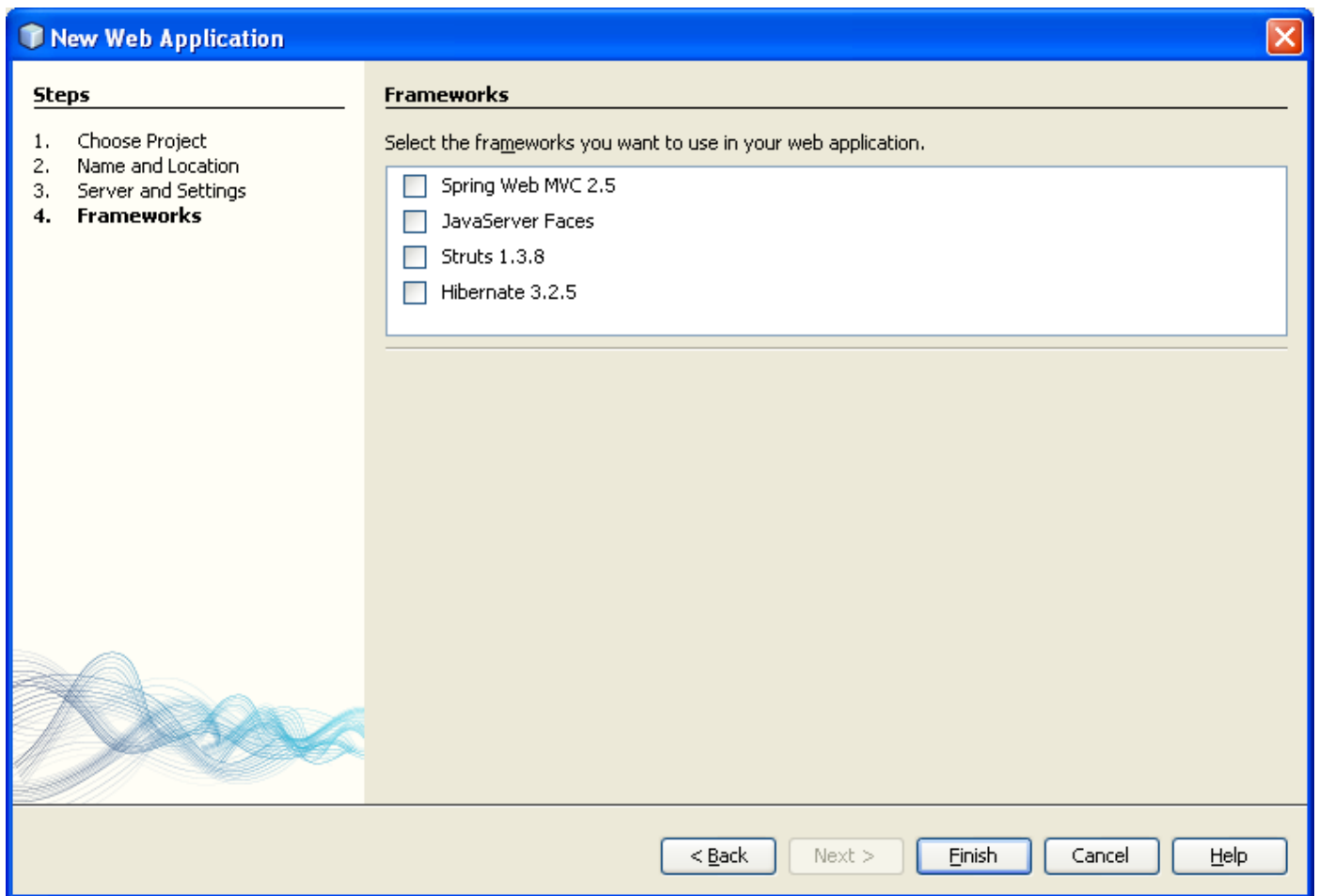
< Back

Next >

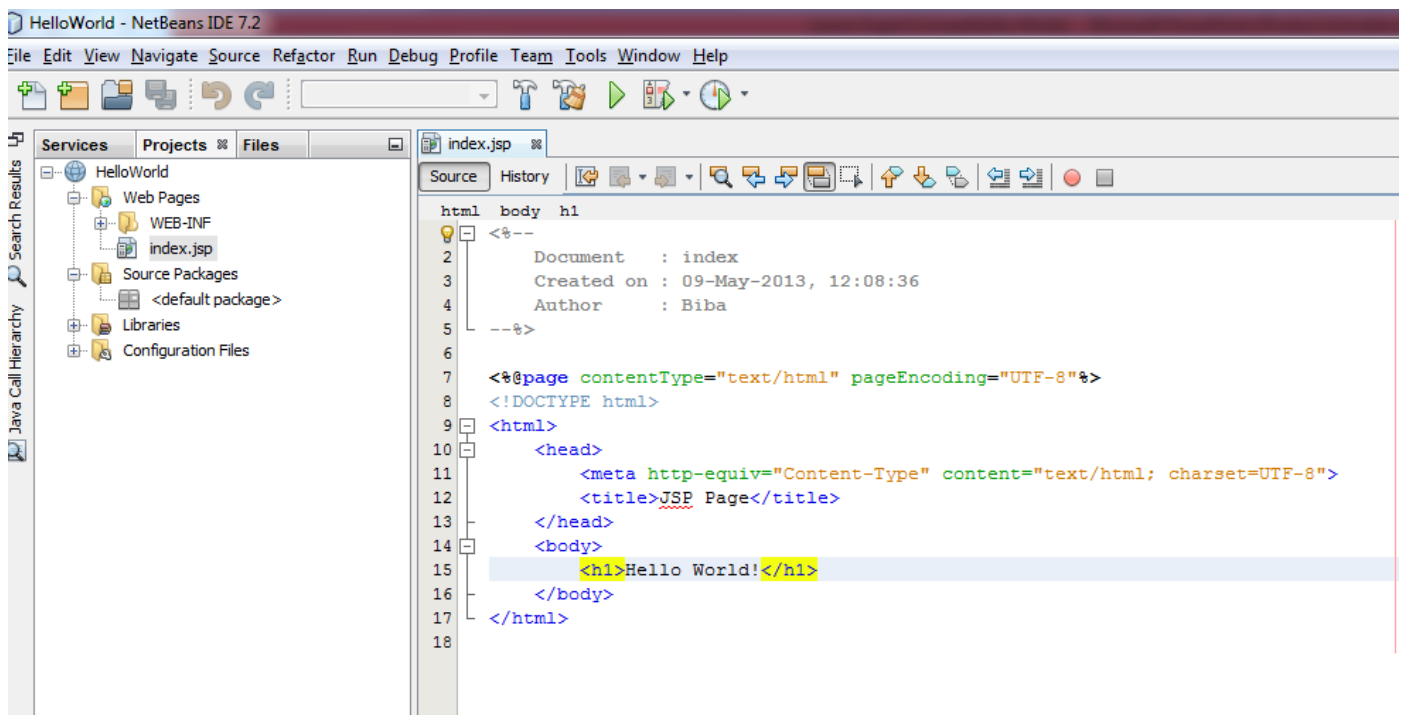
Finish

Cancel

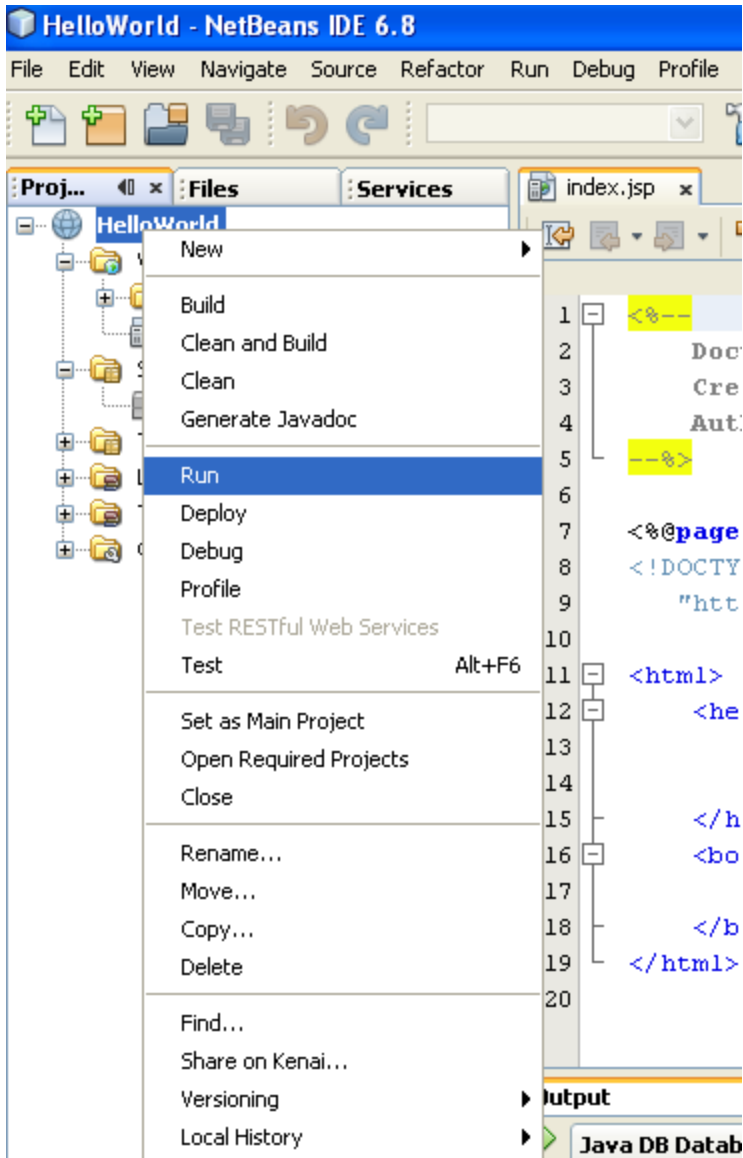
Help



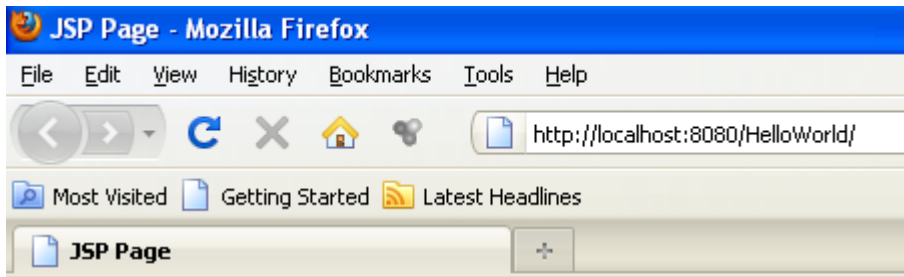
The initial content is this one:



If we run the application:

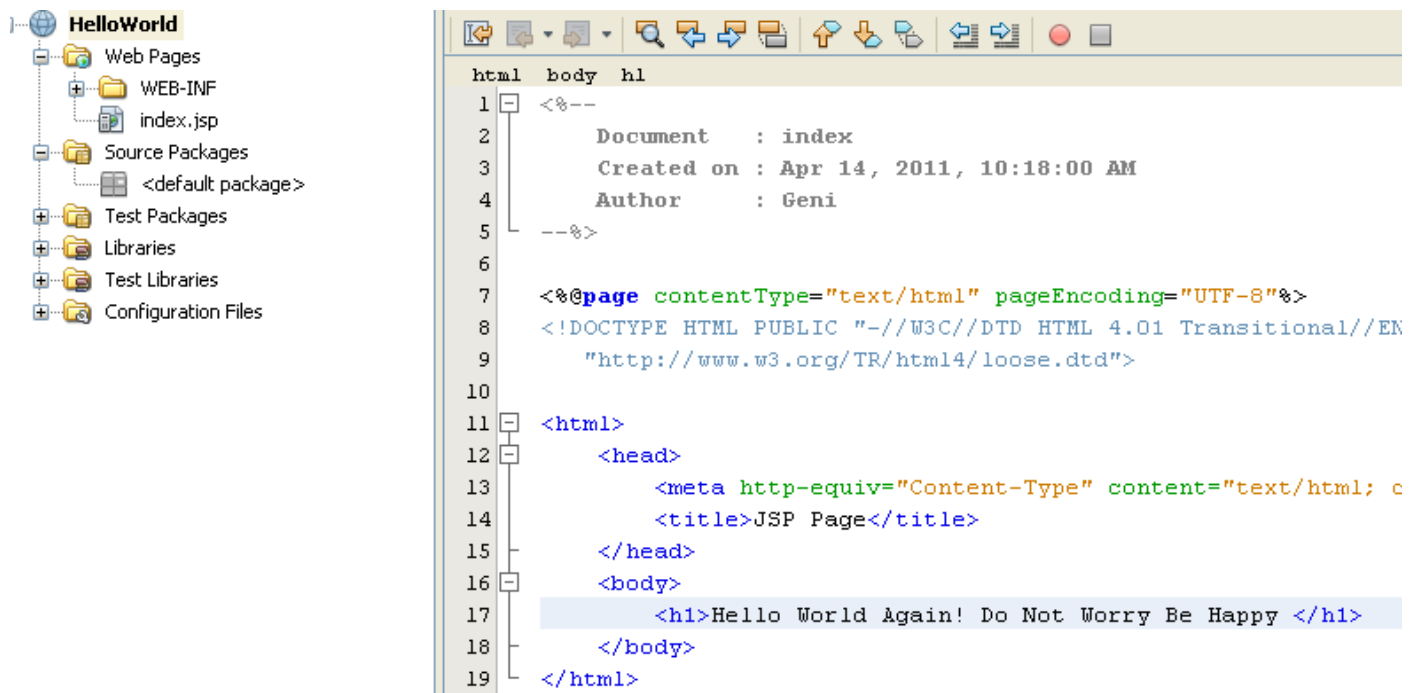


You will see the following:



Hello World!

If we change the text as follows:



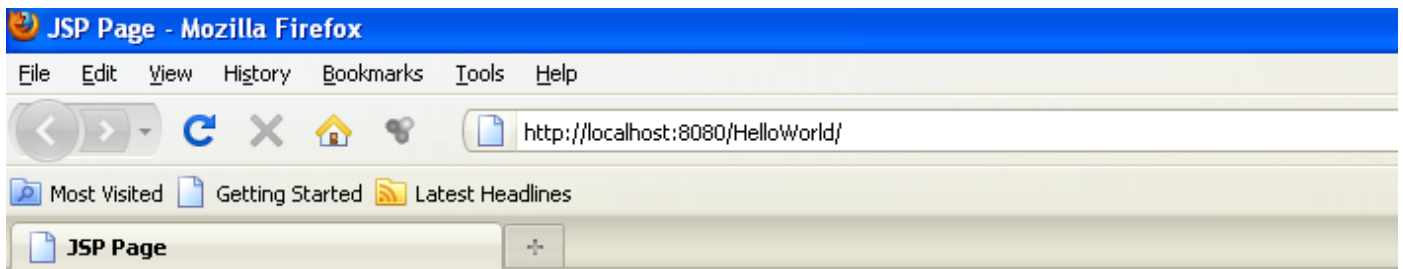
Press Refresh in the browser as follows:



Hello World!

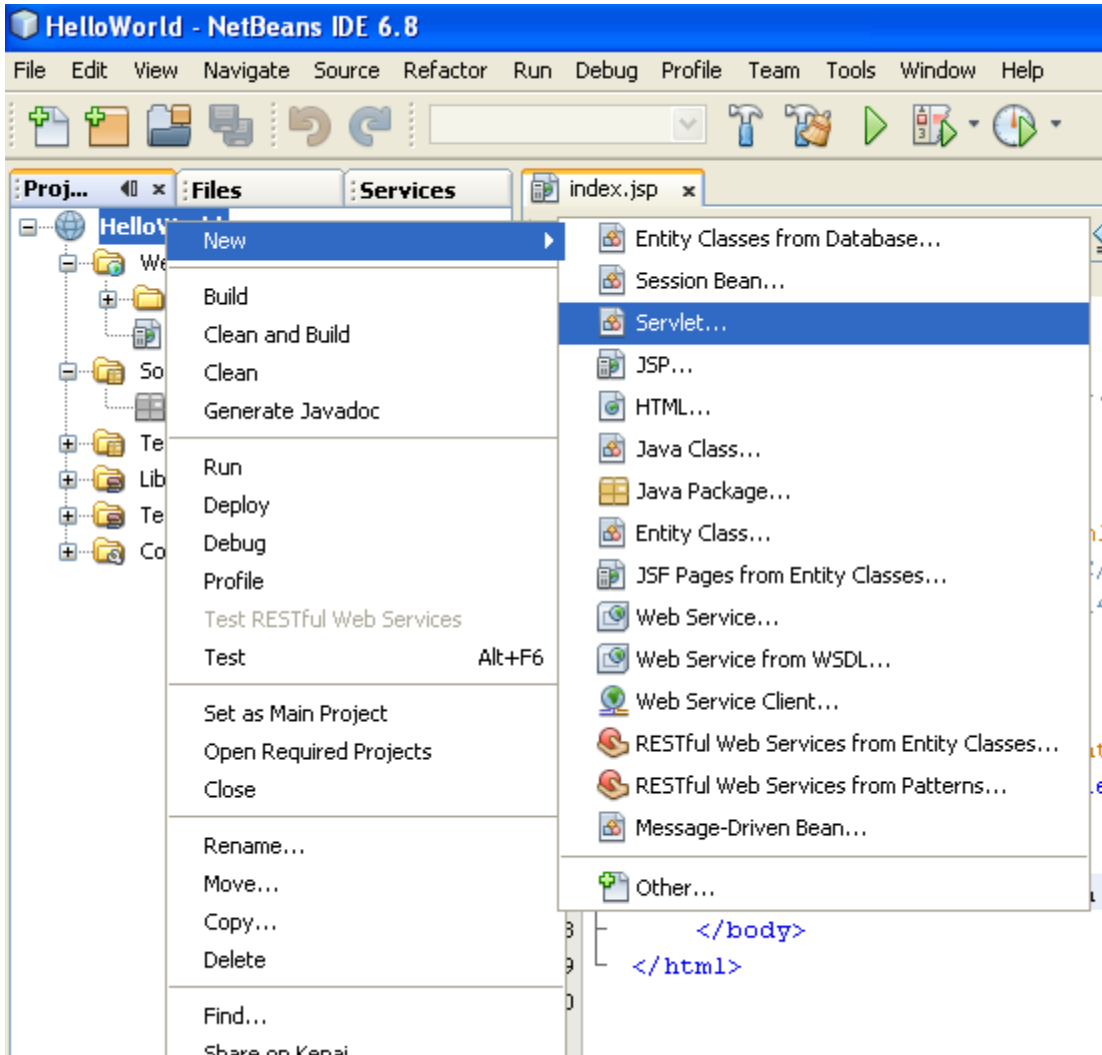
After refreshing the web page you will get:

:



Hello World Again! Do Not Worry Be Happy

Let us now add a servlet to the project:



New Servlet



Steps

1. Choose File Type
2. **Name and Location**
3. Configure Servlet Deployment

Name and Location

Class Name:

Project:

Location: ▼

Package: ▼

Created File:

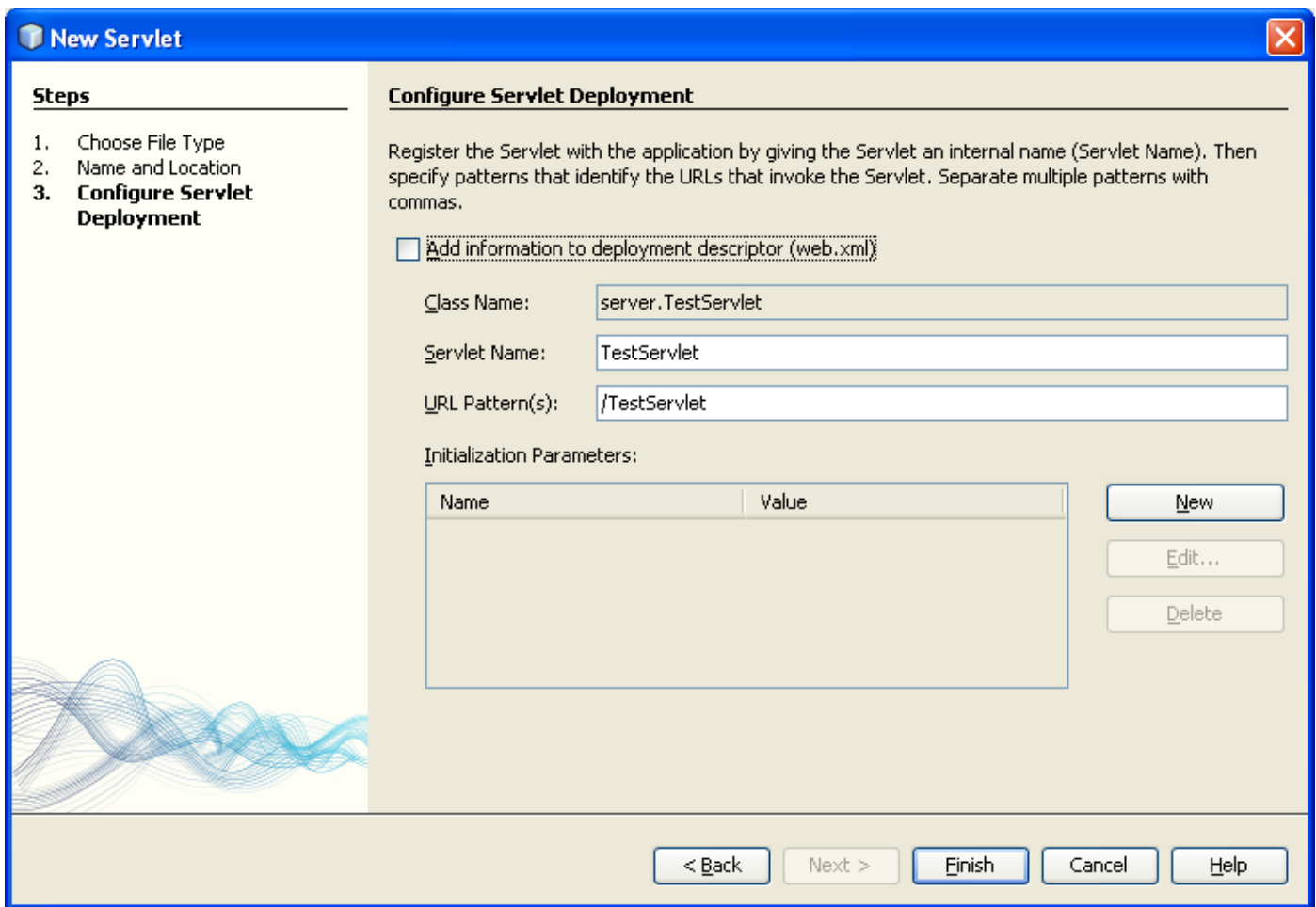
< Back

Next >

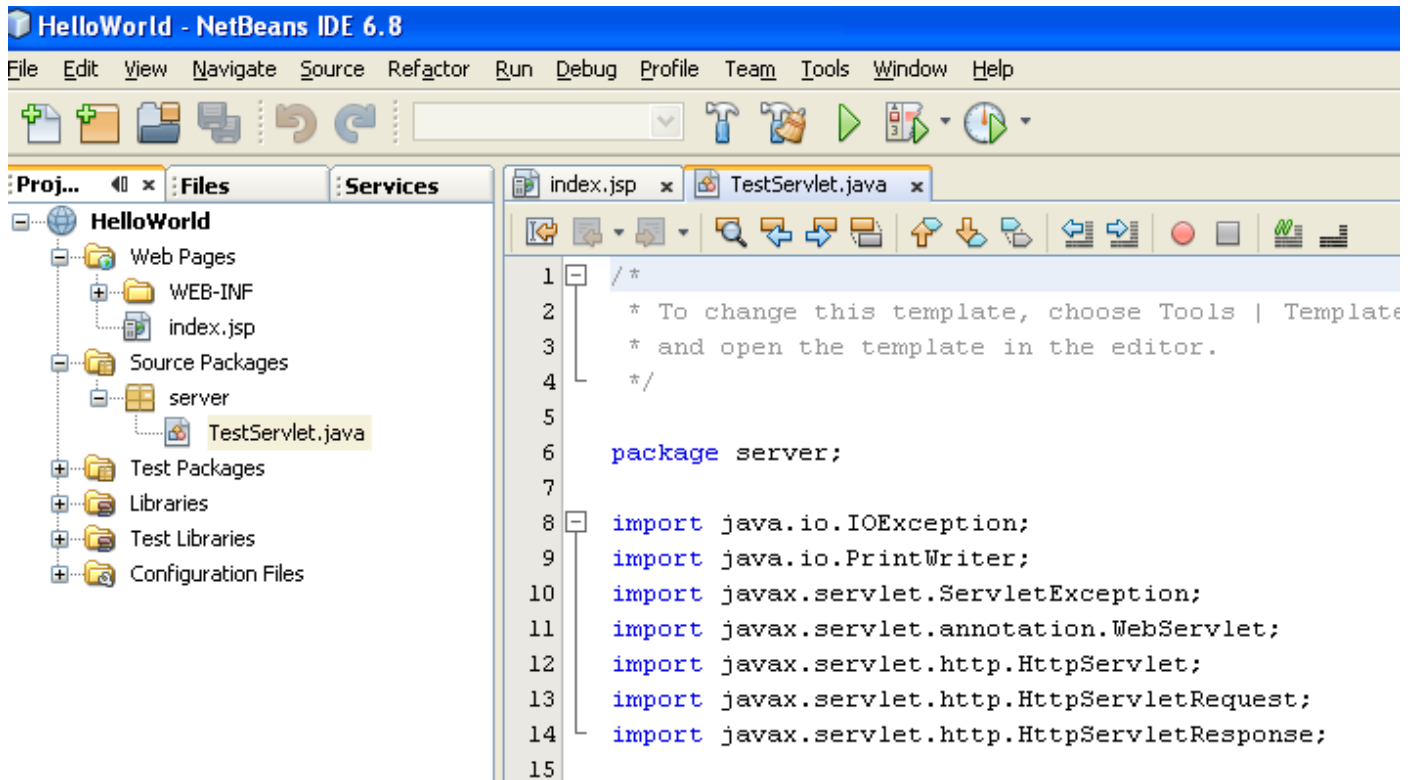
Finish

Cancel

Help



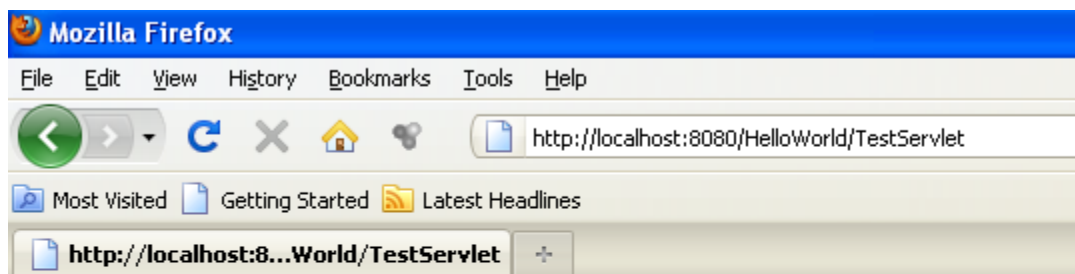
The following is created:



You will see in the Servlet a TODO part generated for you automatically:

```
index.jsp x TestServlet.java x
28     * @throws IOException if an I/O error occurs
29     */
30     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
31     throws ServletException, IOException {
32         response.setContentType("text/html;charset=UTF-8");
33         PrintWriter out = response.getWriter();
34         try {
35             /* TODO output your page here
36             out.println("<html>");
37             out.println("<head>");
38             out.println("<title>Servlet TestServlet</title>");
39             out.println("</head>");
40             out.println("<body>");
41             out.println("<h1>Servlet TestServlet at " + request.getContextPath (
42             out.println("</body>");
43             out.println("</html>");
44             */
45         } finally {
46             out.close();
47         }
48     }
49 }
```

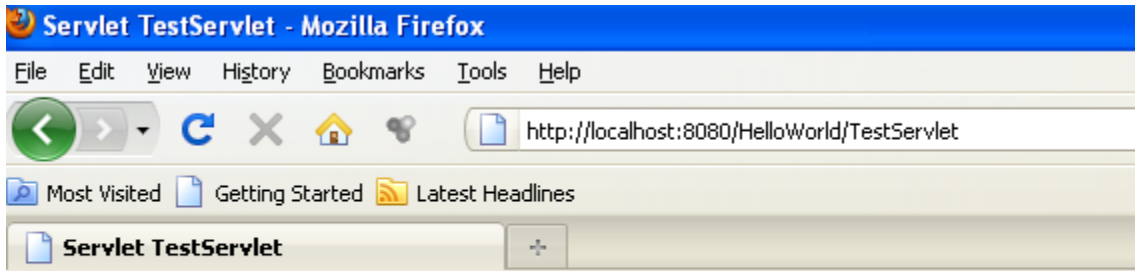
After building and deploying, if you run the application by changing the browser address you will see a blank page:



If you uncomment the TODO part you will get:

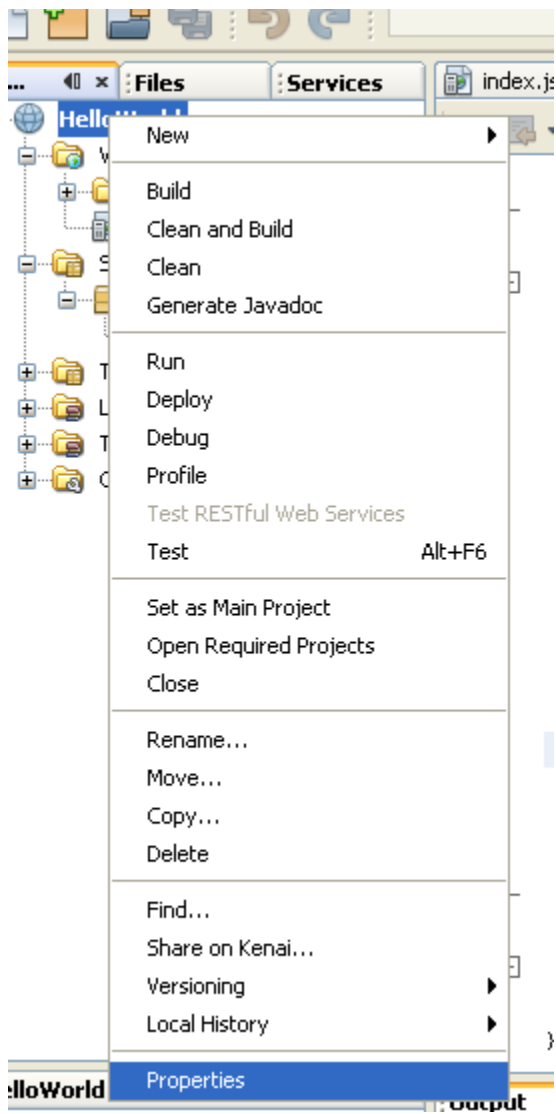
```
*/
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        // TODO output your page here
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet TestServlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet TestServlet at " + request.getContextPath () + "</h1>");
        out.println("</body>");
        out.println("</html>");

    } finally {
        out.close();
    }
}
```

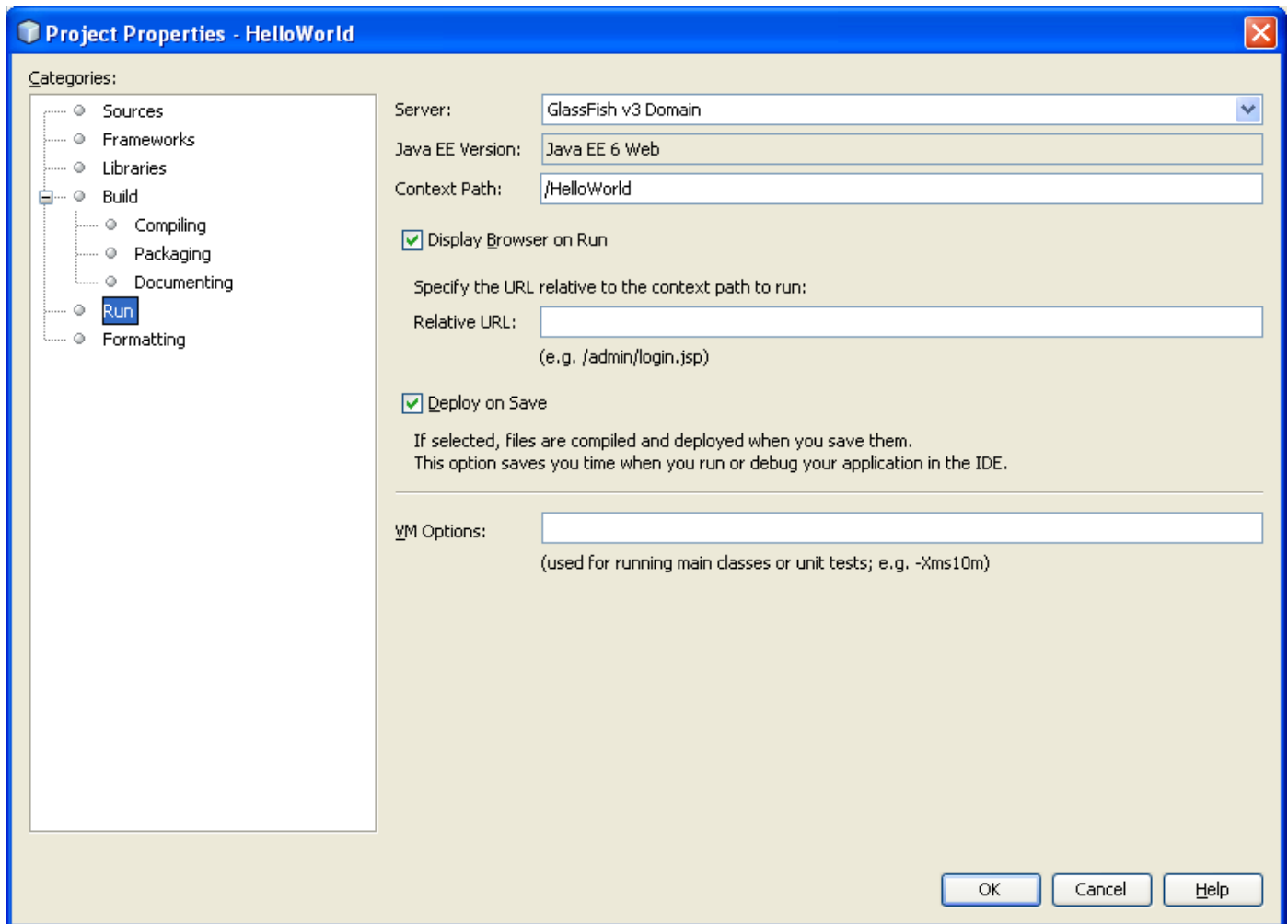


Servlet TestServlet at /HelloWorld

If you want to set deploy on save you can go to Properties:



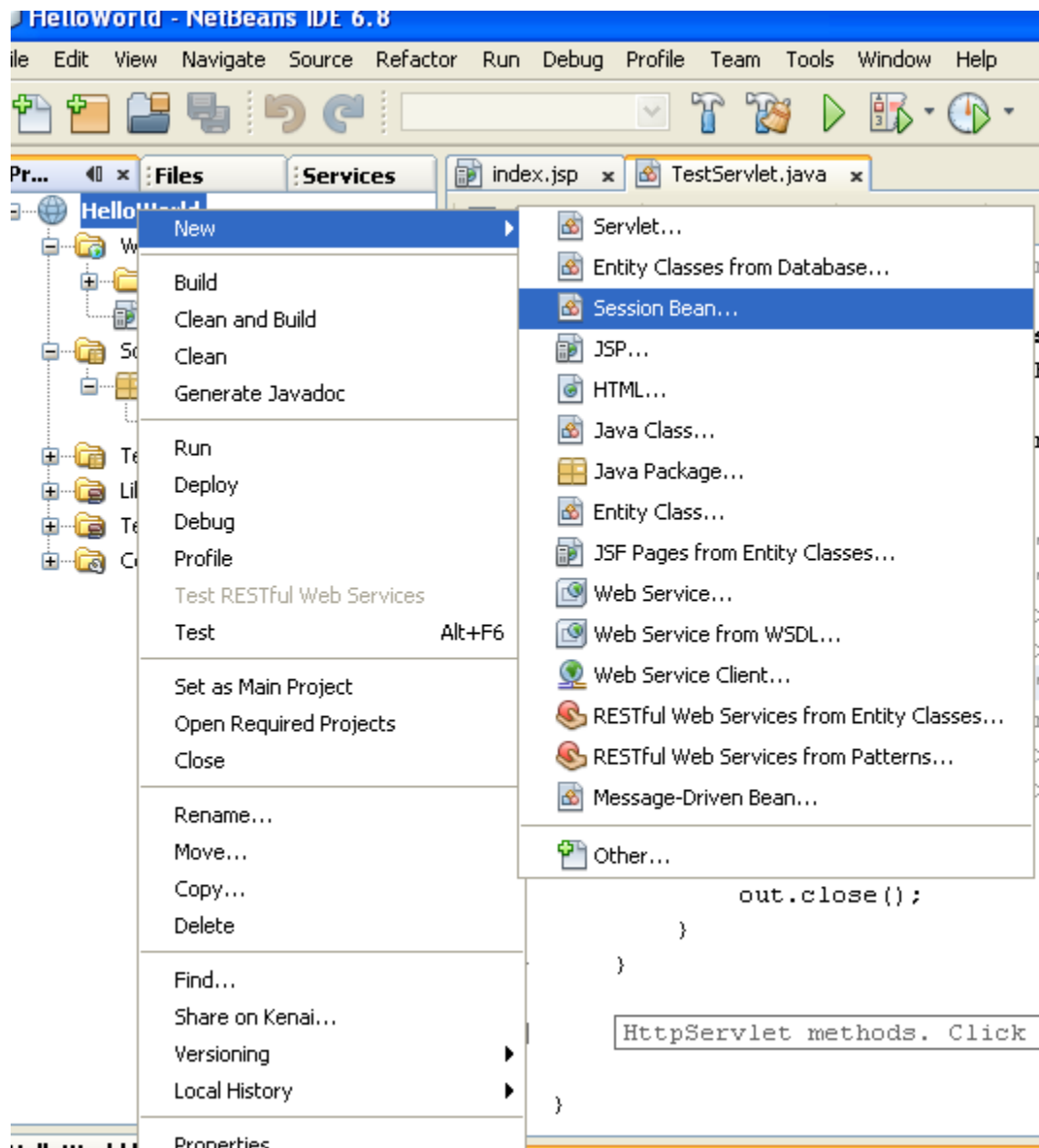
Then Run:

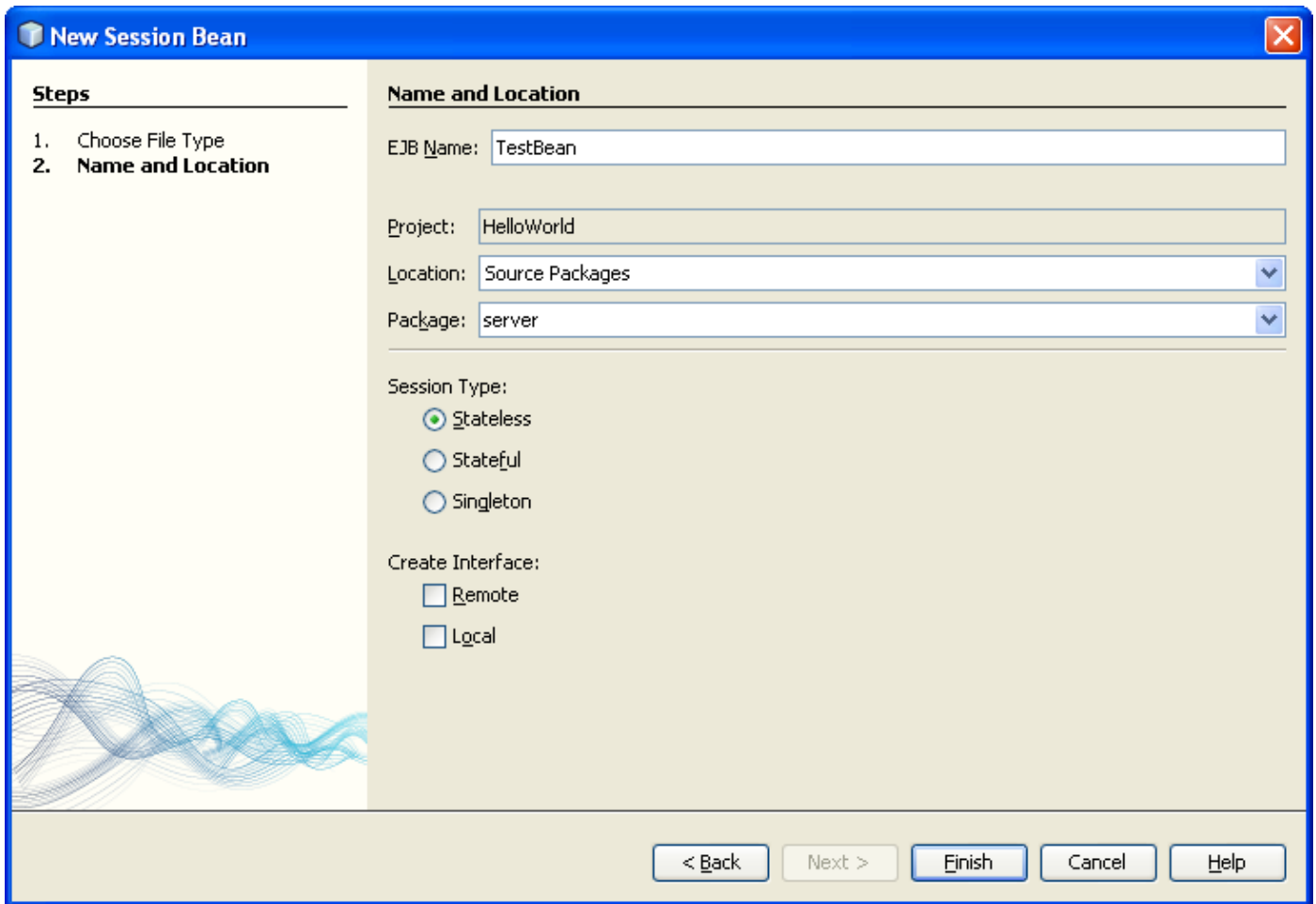


And check Deploy on Save. From now on every time you save the project it is automatically deployed in the server.

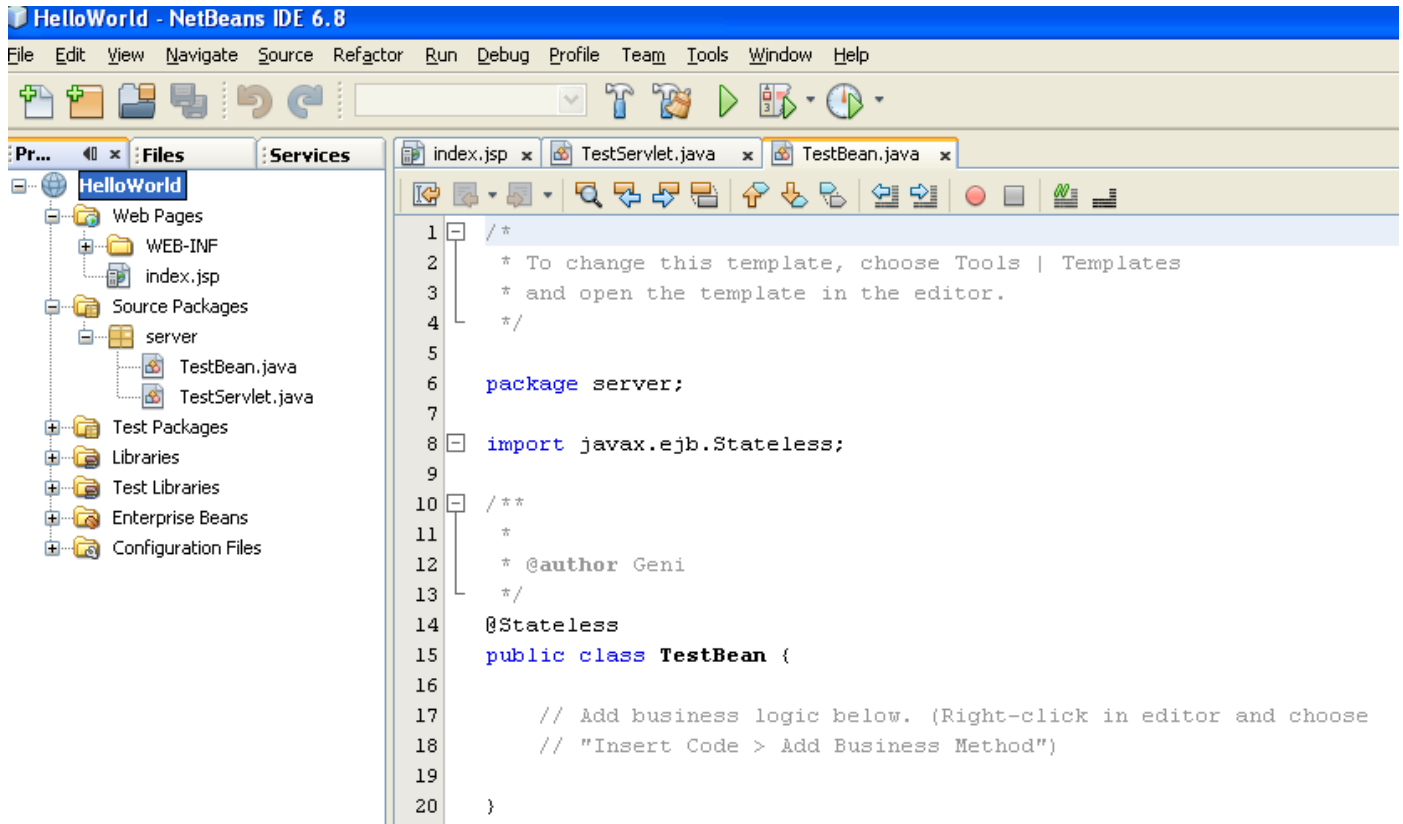
Adding an EJB to the Web Application

Go to New as follows and choose Session Bean:



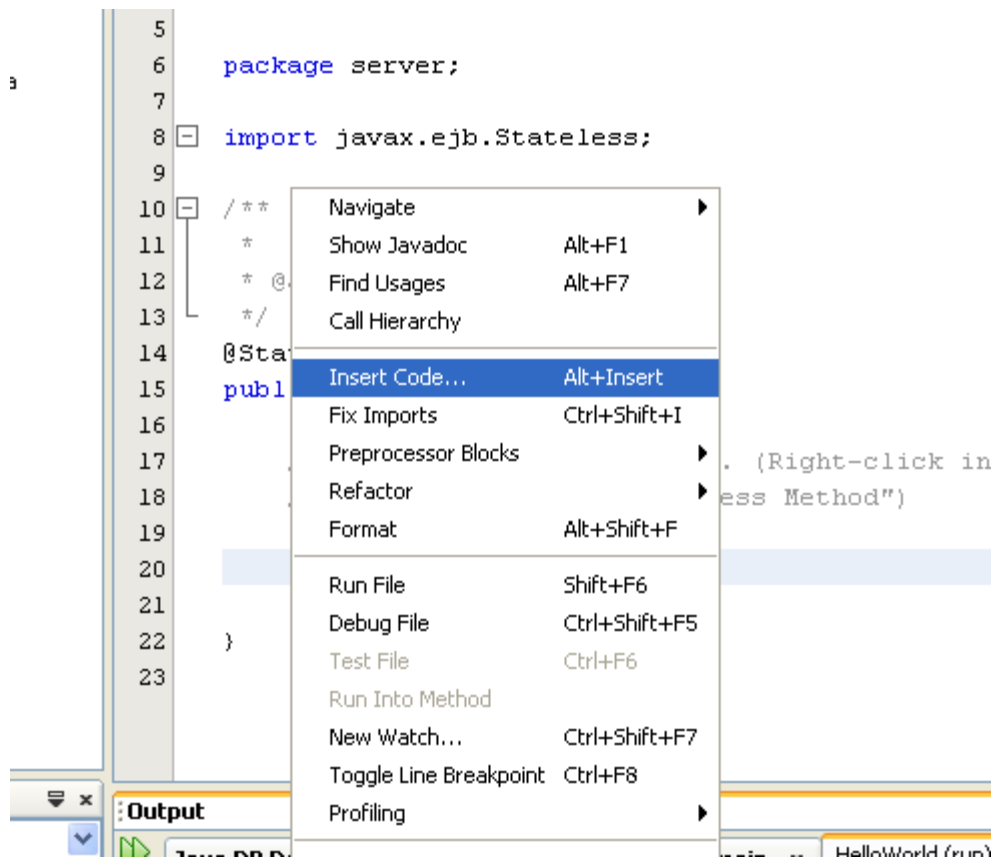


You will see the following:

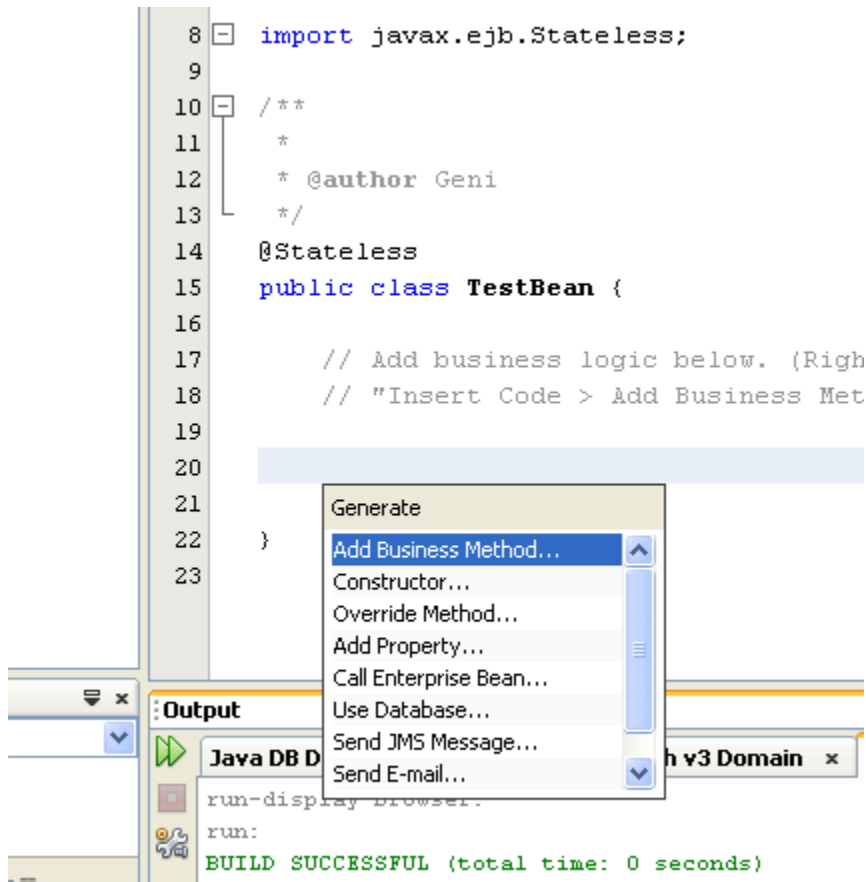


Now lets add some code to the EJB that we created:

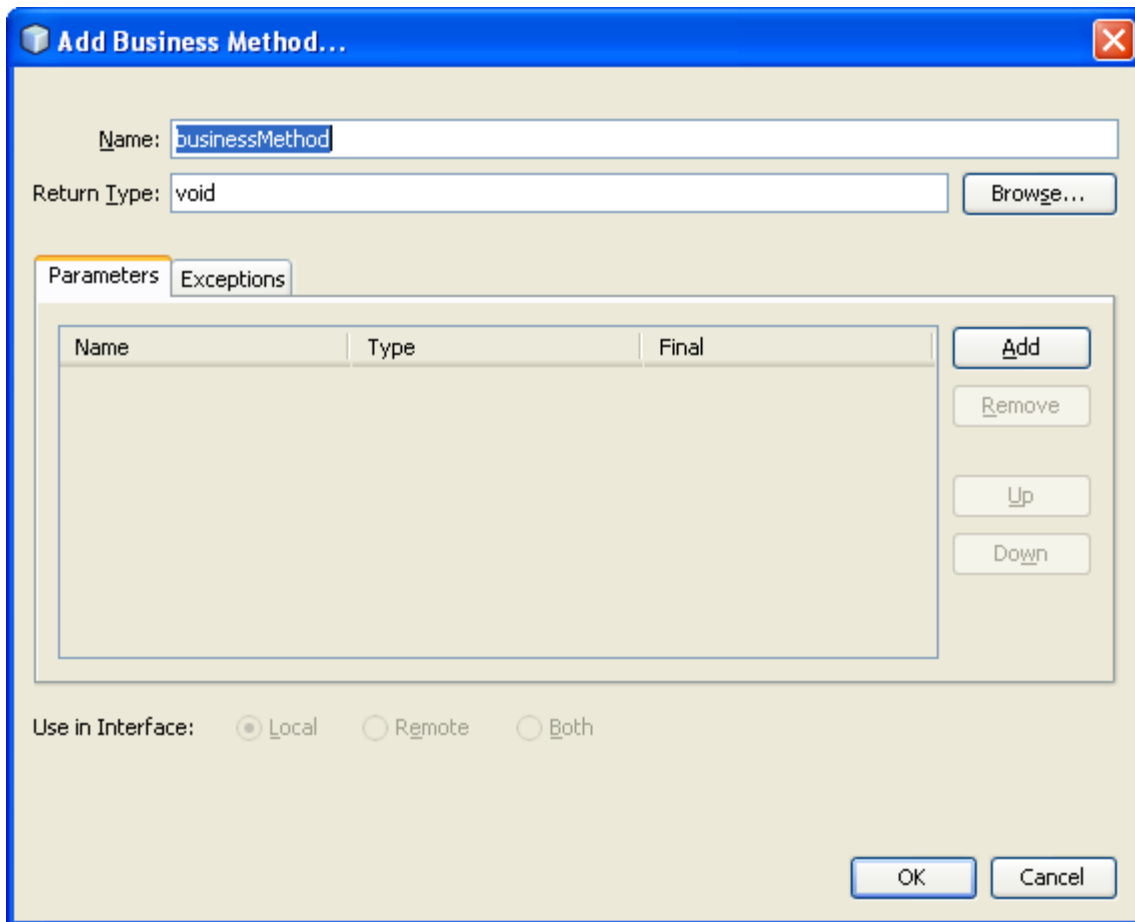
Click with the right of the mouse within the class and choose Insert Code:



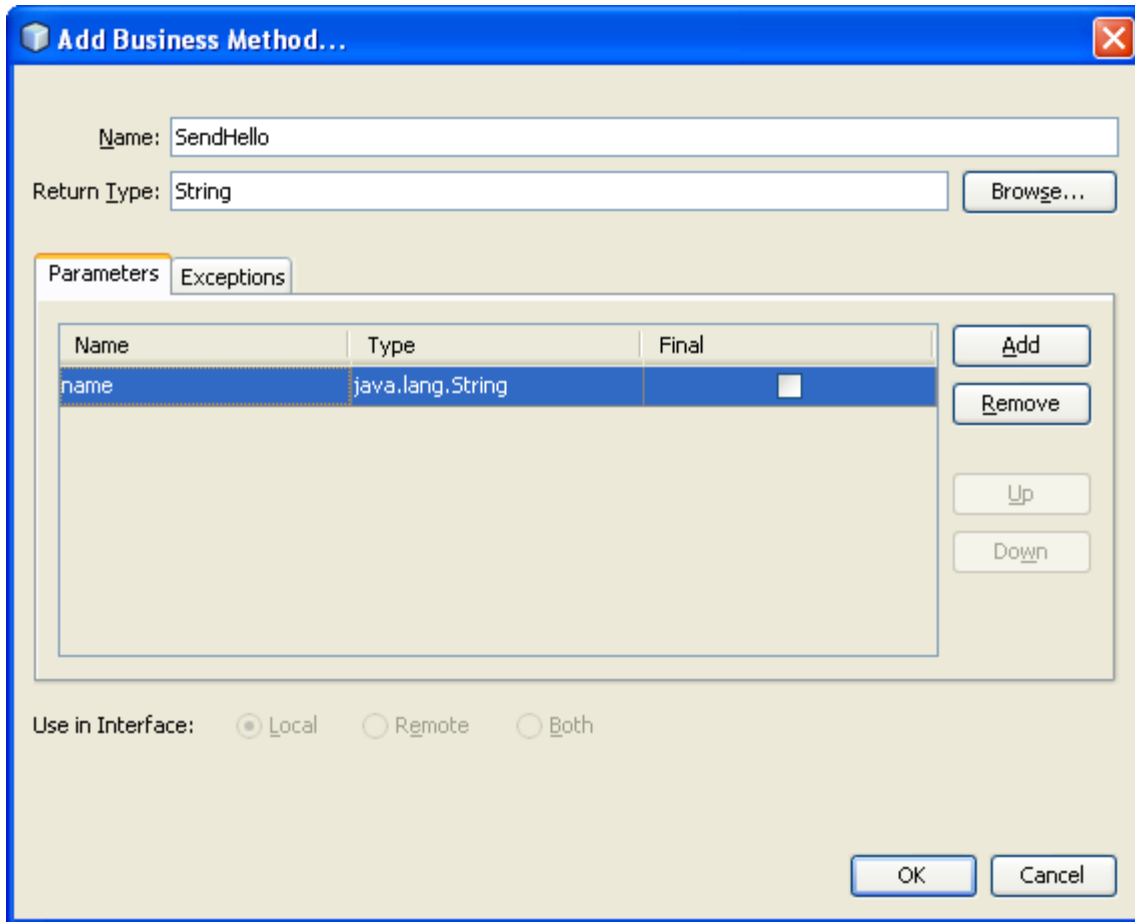
Choose Add Business Method:



The following window will appear:



Write the following in the window:



You will see the following code generated:

```

7
8 import javax.ejb.Stateless;
9
10 /**
11  *
12  * @author Geni
13  */
14 @Stateless
15 public class TestBean {
16
17     public String SendHello(String name) {
18         return null;
19     }
20
21     // Add business logic below. (Right-click in editor and choose
22     // "Insert Code > Add Business Method")
23
24
25
26 }

```

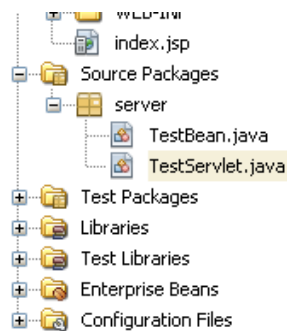
```

13  L   */
14      @Stateless
15      public class TestBean {
16
17      public String SendHello(String name) {
18          return "Name: " + name;
19      }
20
21      // Add business logic below. (Right-click in editor and choose
22      // "Insert Code > Add Business Method")
23
24
25

```

In the class of the servlet add the following code:

@EJB TestBean bean;



```

17      *
18      * @author Geni
19      */
20      @WebServlet(name="TestServlet", urlPatterns={"/TestServlet"})
21      public class TestServlet extends HttpServlet {
22
23          @EJB TestBean bean;
24
25          /**
26           * Processes requests for both HTTP <code>GET</code> and <code>PC
27           * @param request servlet request

```

```

17      *
18      * @author Geni
19      */
20      @WebServlet(name="TestServlet", urlPatterns={"/TestServlet"})
21      public class TestServlet extends HttpServlet {
22
23          @EJB TestBean bean;
24
25          /**
26           * Processes requests for both HTTP <code>GET</code> and <code>PC
27           * @param request servlet request
28           * @param response servlet response
29           * @throws ServletException if a servlet-specific error occurs
30           * @throws IOException if an I/O error occurs

```

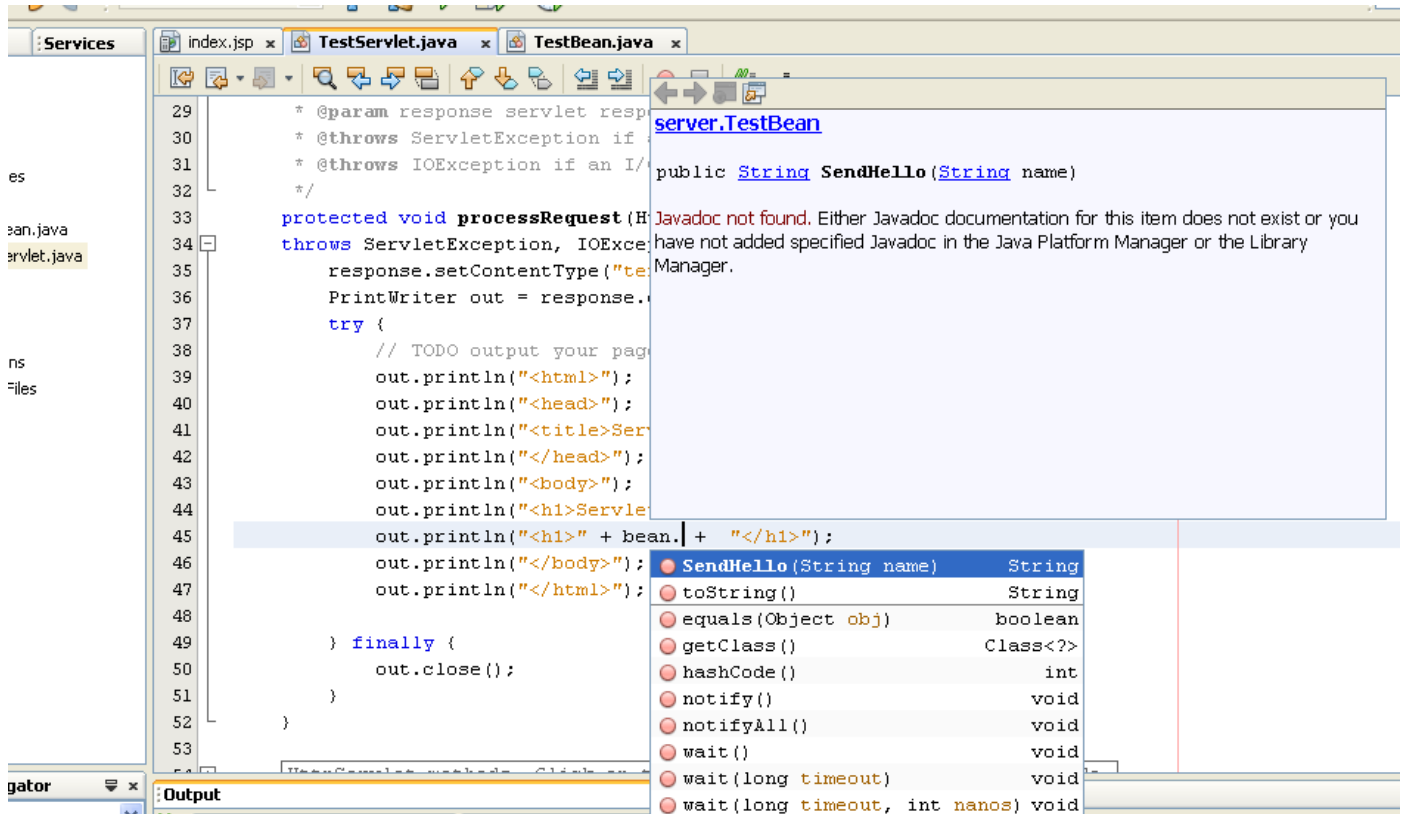
cannot find symbol
symbol: class EJB
location: class server.TestServlet

Surround with ...

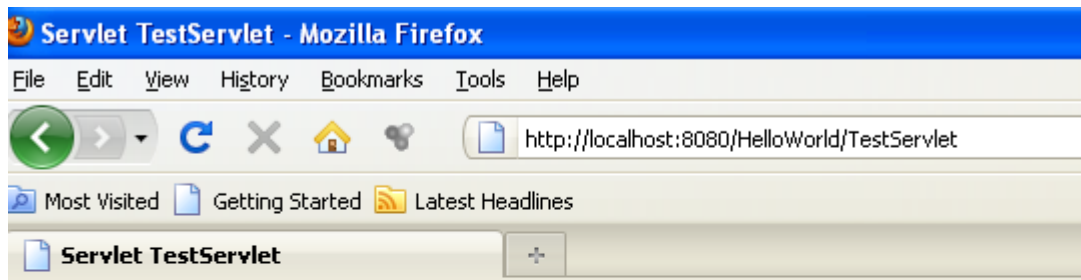
- Add import for javax.ejb.EJB
- Surround with `// <editor-fold defaultstate="collapsed" desc="comment">...`
- Surround with `/*selection*/`

In the code below add:

out.println("<h1>" + bean.SendHello("The Boss") + "</h1>");



When you run the application (In the browser you should add: <http://localhost:8080/HelloWorld/TestServlet>.) or if you just refresh the browser you will get:



Servlet TestServlet at /HelloWorld

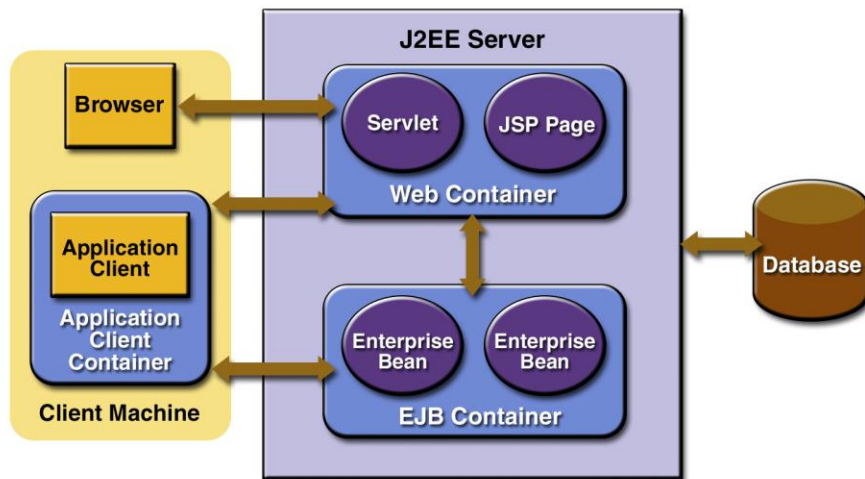
Name: The Boss

We have completed the application and shown how a servlet can be connected with an Enterprise Java Bean.

4. Developing a web application with a Servlet and an EJB that connects to a database.

Following the schema below, now we will try to build an application where the EJB is connected to a database.

J2EE

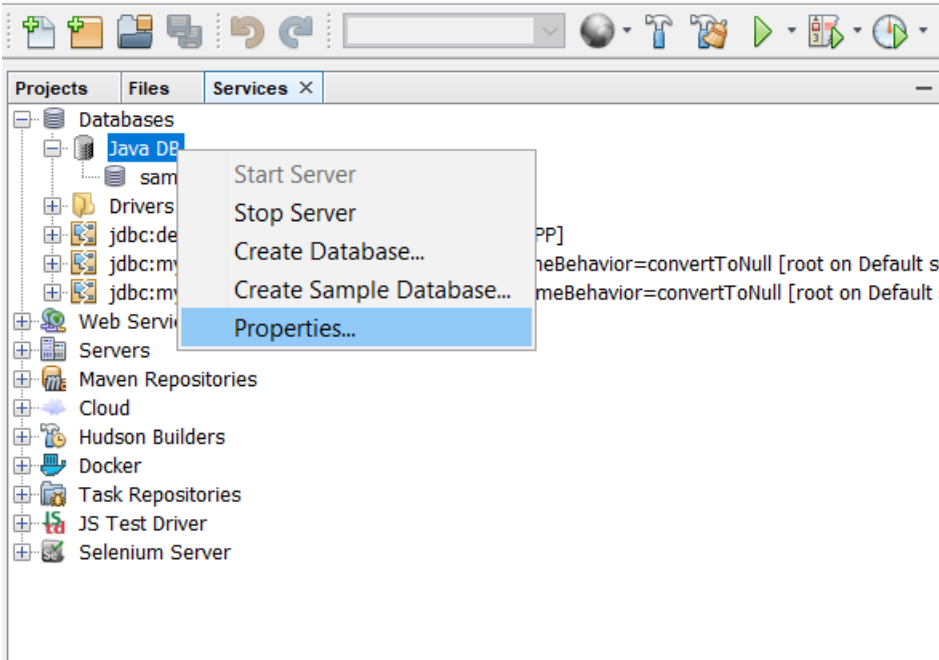


Take the project that we developed in Section 3 and perform the following operations.

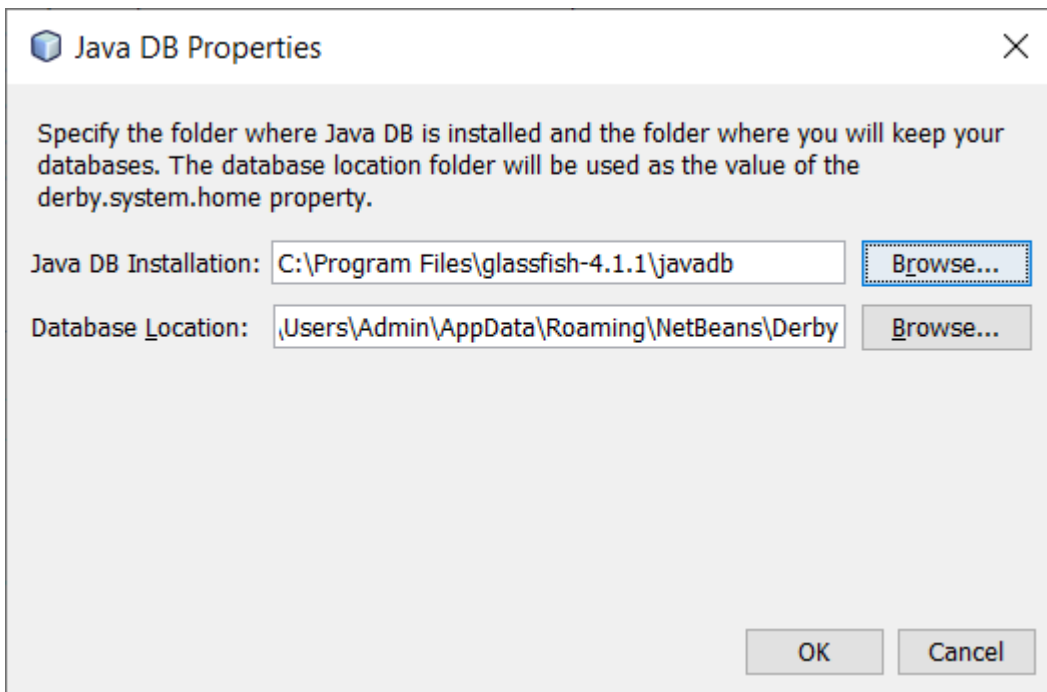
First of all change the properties of the Java DB as follows:

NetBeans IDE 8.2

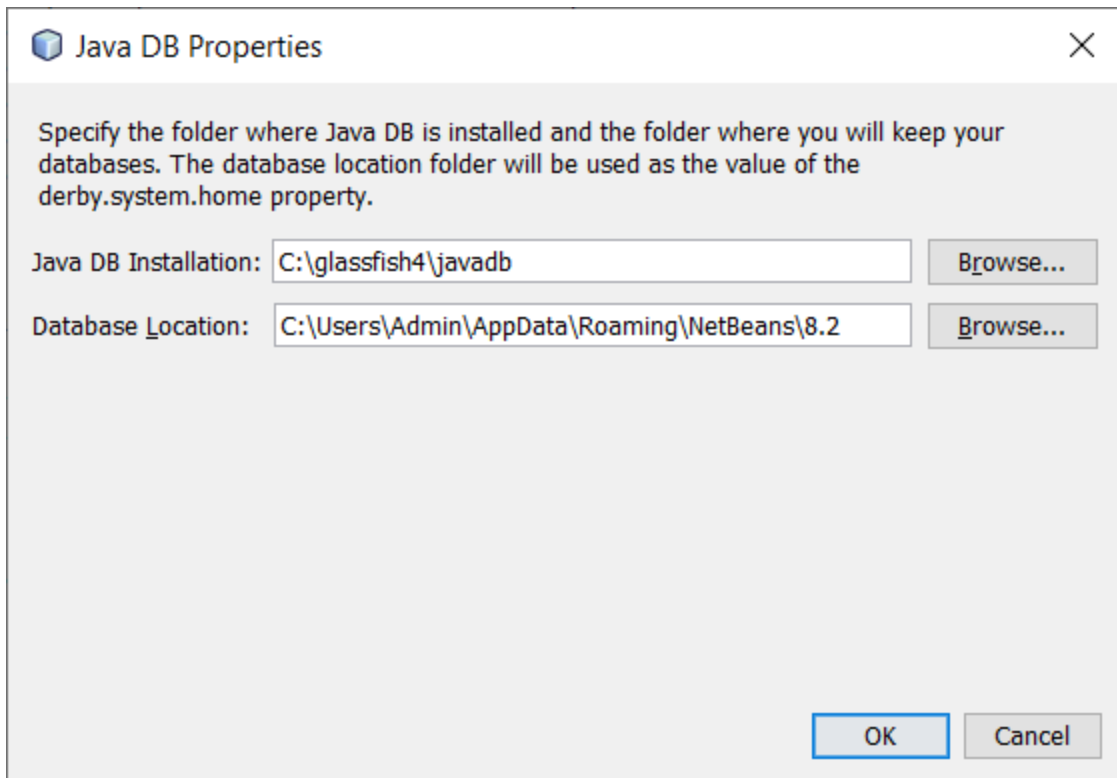
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window I



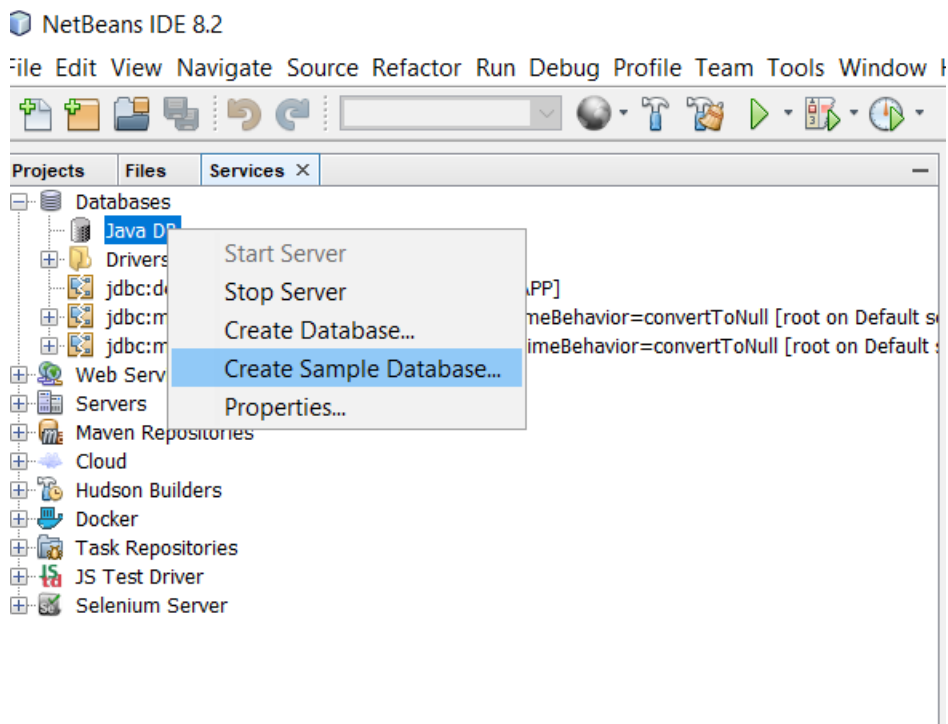
Open Properties:



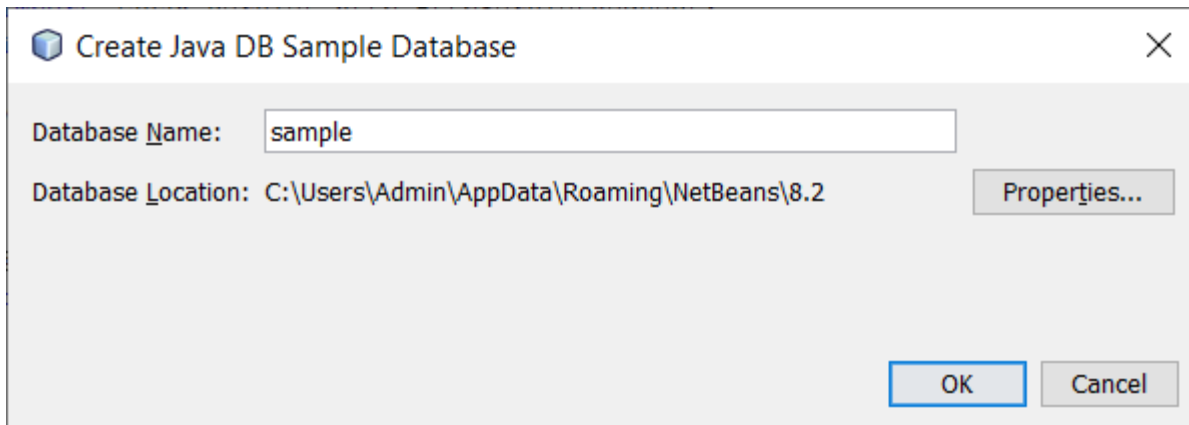
Change the above fields as follows:



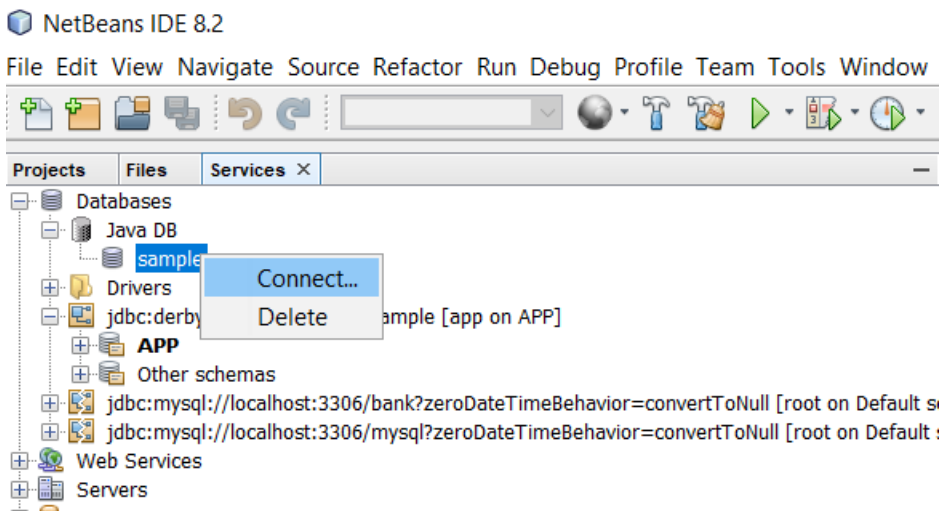
Then create a sample DB:



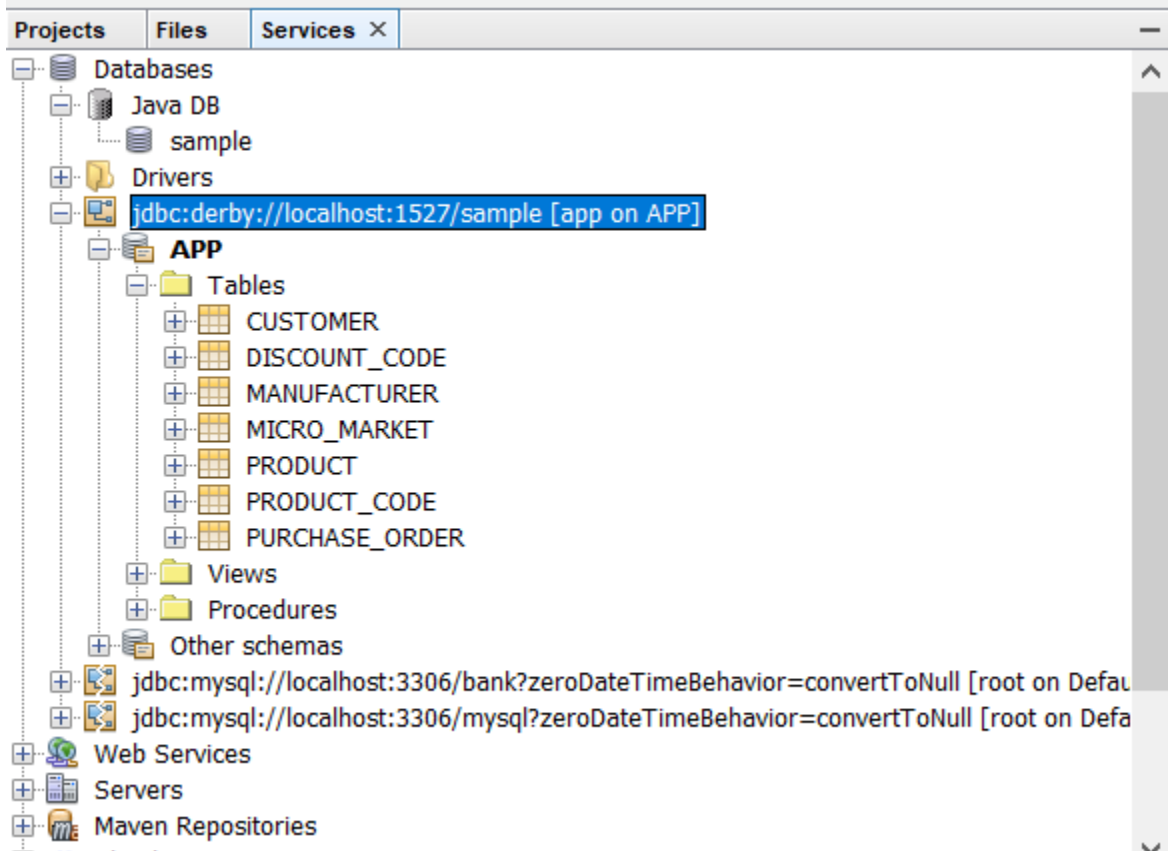
Name the database as “sample” as follows:



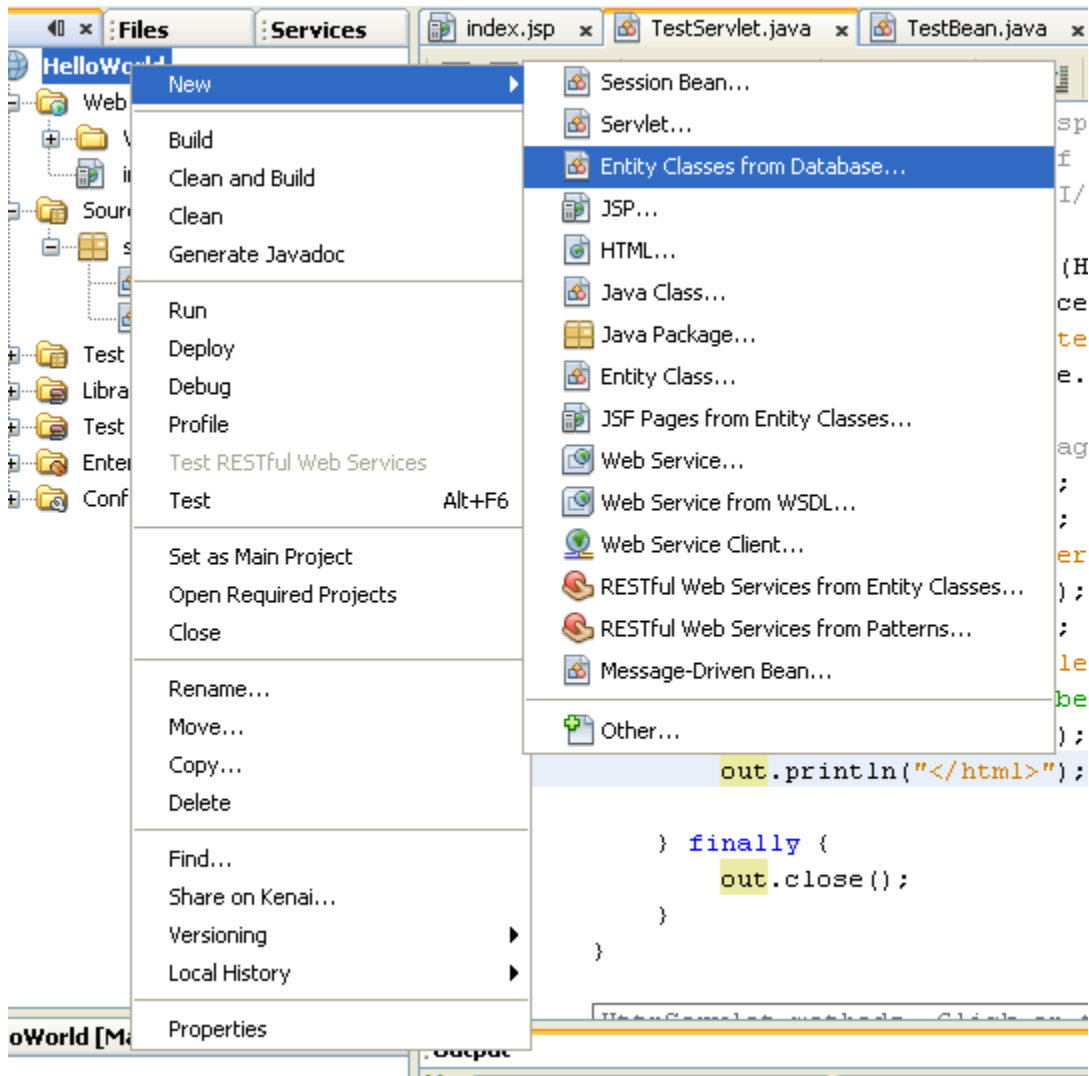
Then connect the database as follows:



You can now open the database as follows:

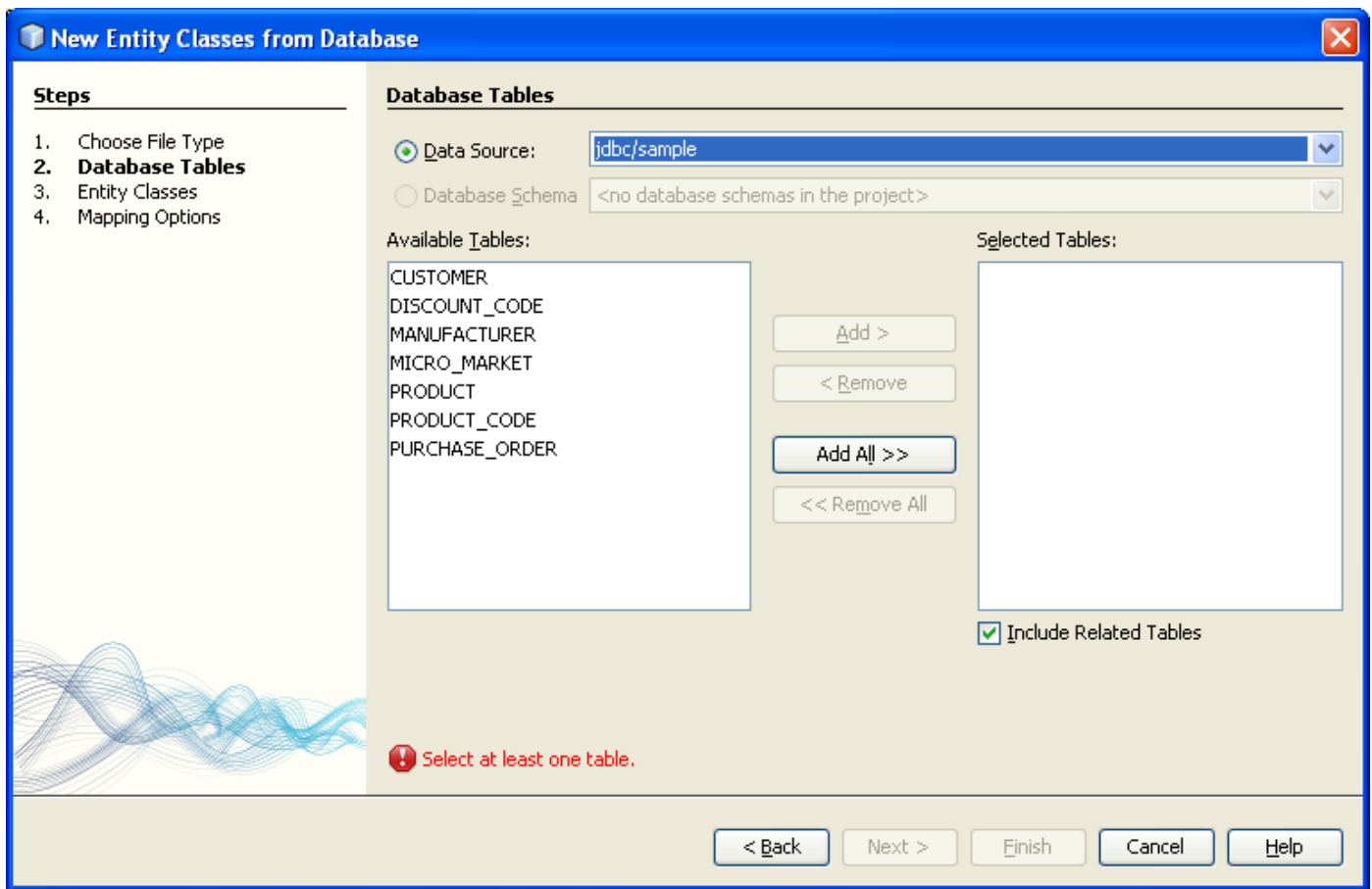


Then create an Entity EJB as follows:



In the window below, choose JDBC/Sample:

:



From the table select one table, for example Customer:

New Entity Classes from Database

- Steps**
1. Choose File Type
 - 2. Database Tables**
 3. Entity Classes
 4. Mapping Options

Database Tables

Data Source: jdbc/sample

Database Schema: <no database schemas in the project>

Available Tables:

MANUFACTURER
MICRO_MARKET
PRODUCT
PRODUCT_CODE
PURCHASE_ORDER

Add >

< Remove

Add All >>

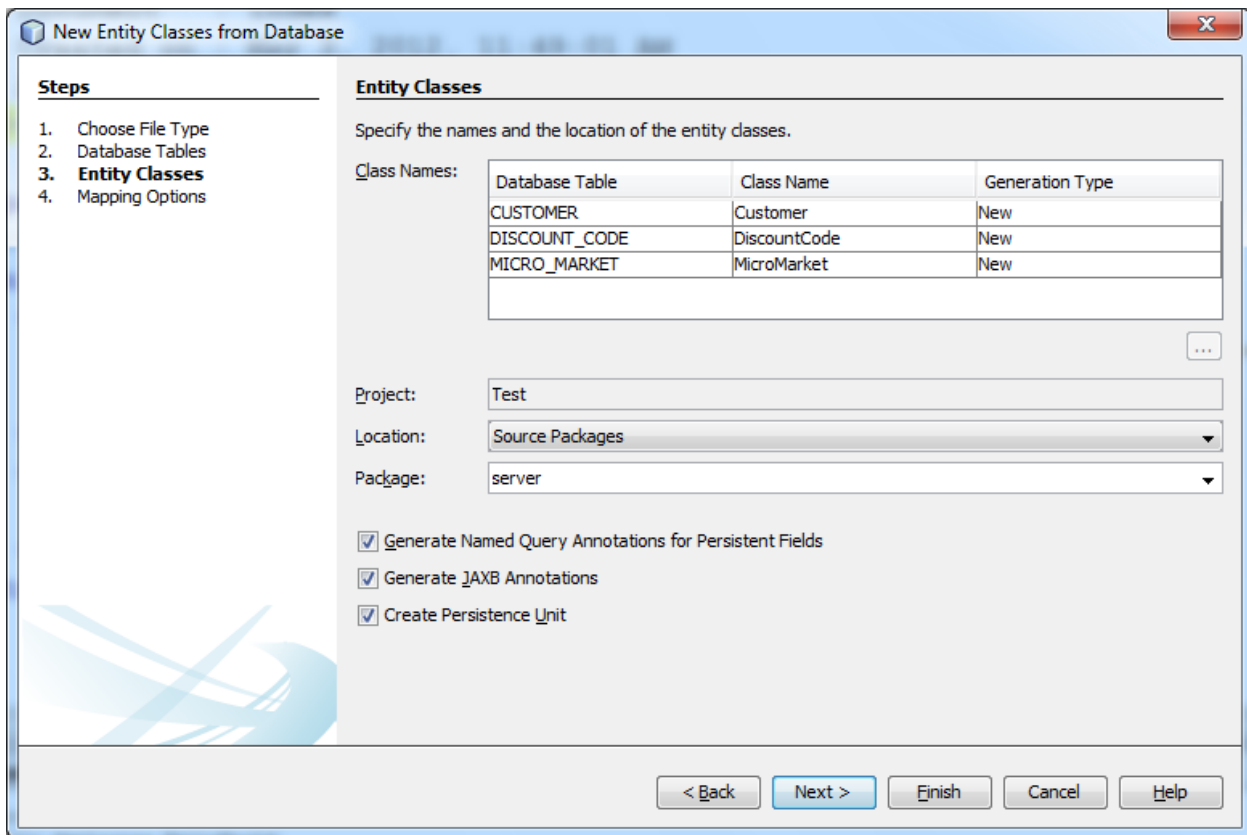
<< Remove All

Selected Tables:

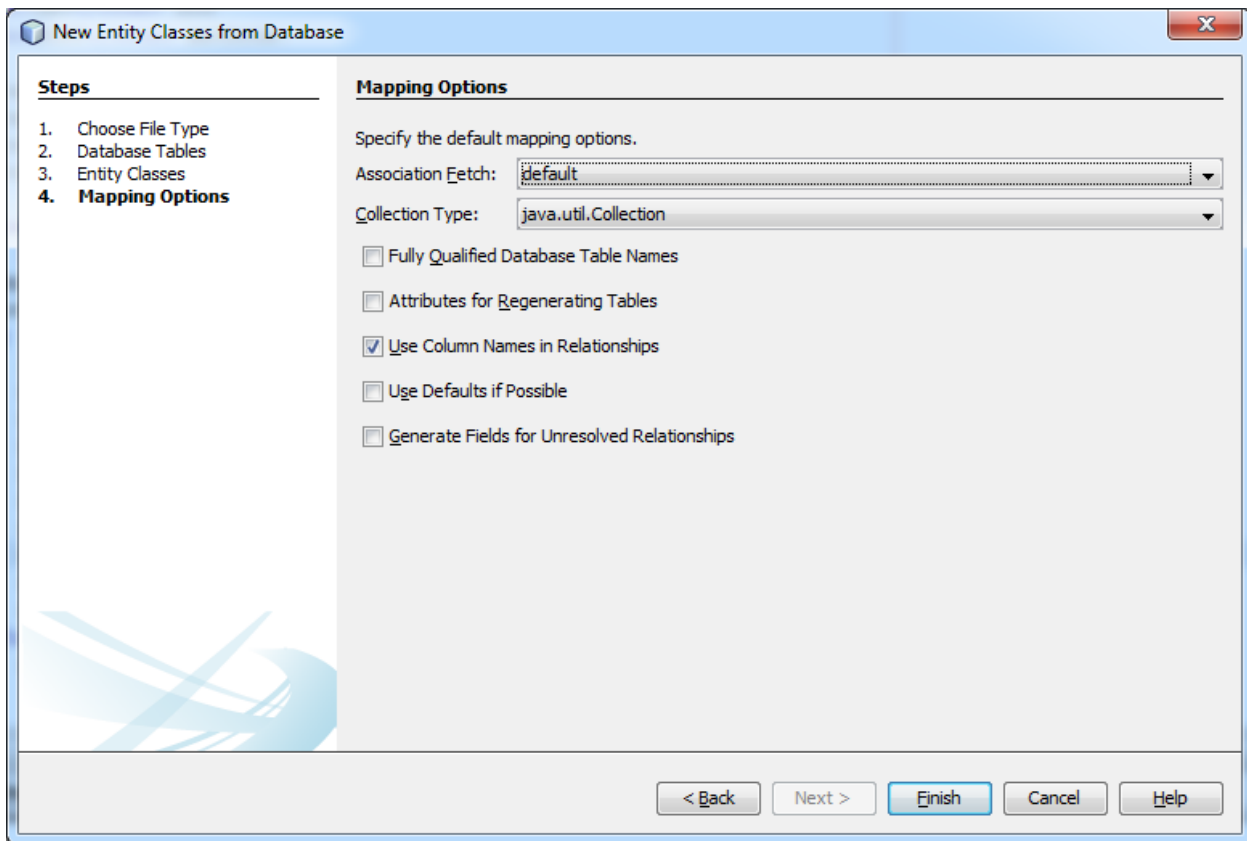
CUSTOMER
DISCOUNT_CODE

Include Related Tables

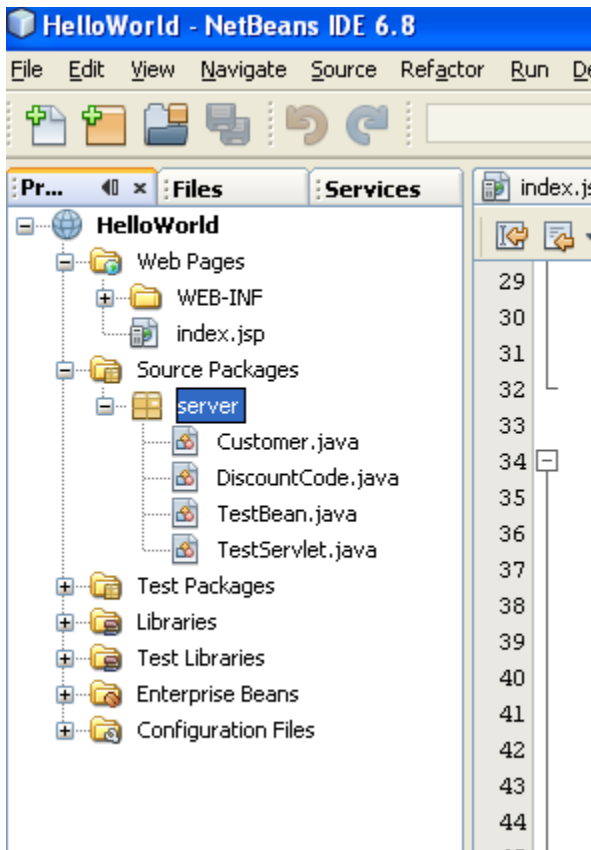
< Back Next > Finish Cancel Help



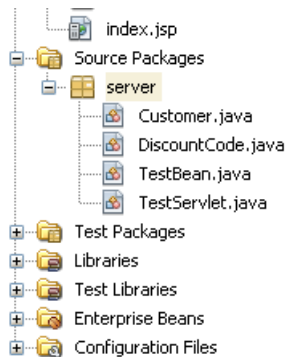
Click "Next".



Click "Finish".



In the servlet add the following code:

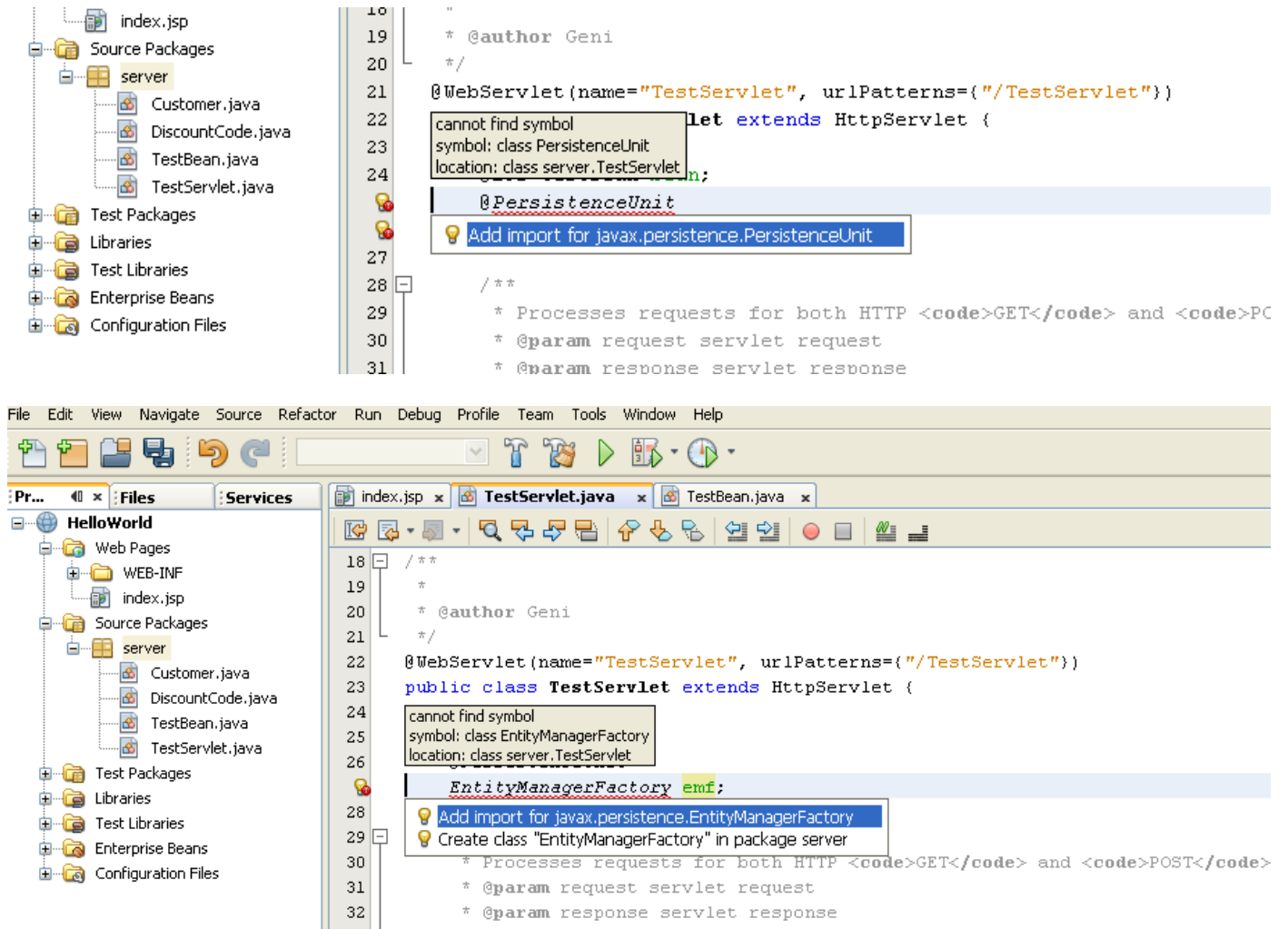


```

18      *
19      * @author Geni
20      */
21      @WebServlet(name="TestServlet", urlPatterns={"/TestServlet"})
22      public class TestServlet extends HttpServlet {
23
24          @EJB TestBean bean;
25          @PersistenceUnit
26          EntityManagerFactory emf;
27
28          /**
29           * Processes requests for both HTTP <code>GET</code> and <code>POST</code>
30           * @param request servlet request

```

Add imports:



Now in the code we should call the EJB. Perform the following code changes in the servlet:

NetBeans IDE 8.0

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

index.jsp x TestServlet.java x TestBean.java

javax.persistence.EntityManagerFactory

```

public EntityManager createEntityManager ()
Create a new application-managed EntityManager. This method returns a new
EntityManager instance each time it is invoked. The isOpen method will return true
on the returned instance.

Returns:
entity manager instance

Throws:
IllegalStateException - if the entity manager factory has been closed

```

```

37 protected void processRequest(
38 throws ServletException,
39 response.setContentType("text/html");
40 PrintWriter out = response.getWriter();
41 try {
42 // TODO output your page here
43 out.println("<html>");
44 out.println("<head>");
45 out.println("<title>Servlet TestServlet");
46 out.println("</head>");
47 out.println("<body>");
48 out.println("<h1>Servlet TestServlet at " + bean.SendHello("The"));
49 out.println("</h1>");
50 Customer cust = emf.createEntityManager().find("Customer.findAll");
51 out.println("</body>");
52 out.println("</html>");
53
54 } finally {
55 out.close();
56 }
57 }
58
59 HttpServlet methods. Click on the + sign on the left to expand the list.
94

```

```

40 PrintWriter out = response.getWriter();
41 try {
42 // TODO output your page here
43 out.println("<html>");
44 out.println("<head>");
45 out.println("<title>Servlet TestServlet");
46 out.println("</head>");
47 out.println("<body>");
48 out.println("<h1>Servlet TestServlet at " + bean.SendHello("The"));
49 out.println("</h1>");
50 Customer cust = emf.createEntityManager().find("Customer.findAll");
51 out.println("</body>");
52 out.println("</html>");
53
54 } finally {
55 out.close();
56 }
57 }
58
59 HttpServlet methods. Click on the + sign on the left to expand the list.
94
95 }

```

```

Parameters:
name - the name of a query defined in metadata

Returns:
the new query instance

Throws:
java.lang.IllegalArgumentException - if a query has not been defined
with the given name or if the query string is found to be invalid

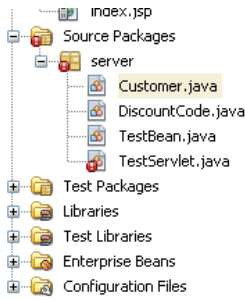
```

```

clear () void
close () void
contains (Object entity) boolean
createNamedQuery (String name) Query
createNamedQuery (String name, Class<T> resultClass) TypedQuery<T>
createNativeQuery (String sqlString) Query
createNativeQuery (String sqlString, Class resultClass) Query
createNativeQuery (String sqlString, String resultSetMapping) TypedQuery<T>
createQuery (CriteriaQuery<T> criteriaQuery) TypedQuery<T>
createQuery (String qlString) Query
createQuery (String qlString, Class<T> resultClass) TypedQuery<T>
detach (Object entity) void

```

From Customer take a query for example "Customer.findAll":

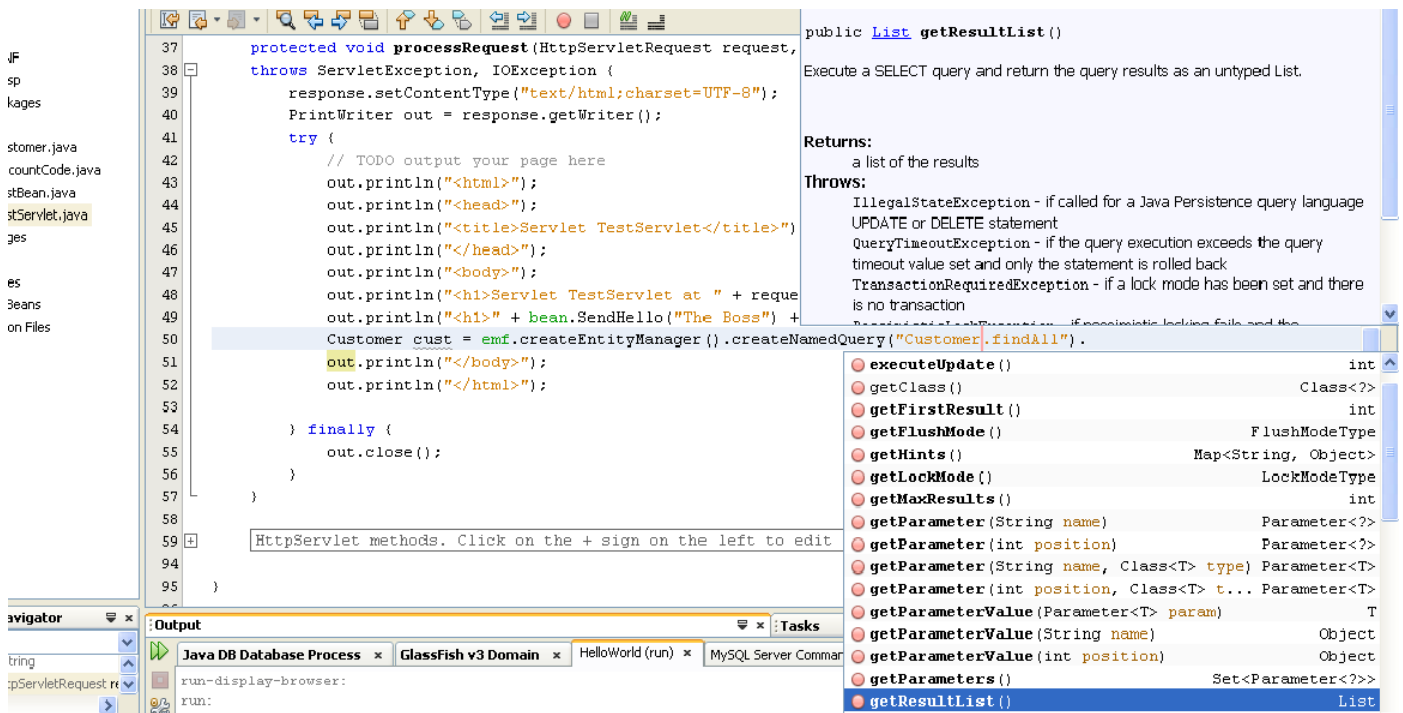


```

15 import javax.persistence.NamedQueries;
16 import javax.persistence.NamedQuery;
17 import javax.persistence.Table;
18
19 /**
20  *
21  * @author Geni
22  */
23 @Entity
24 @Table(name = "CUSTOMER")
25 @NamedQueries({
26     @NamedQuery(name = "Customer.findAll", query = "SELECT c FROM Customer c"),
27     @NamedQuery(name = "Customer.findById", query = "SELECT c FROM Customer c WHERE c.id = :id"),
28     @NamedQuery(name = "Customer.findByZip", query = "SELECT c FROM Customer c WHERE c.zip = :zip"),
29     @NamedQuery(name = "Customer.findByName", query = "SELECT c FROM Customer c WHERE c.name = :name"),
30     @NamedQuery(name = "Customer.findByAddressline1", query = "SELECT c FROM Customer c WHERE c.addressline1 = :addressline1"),
31     @NamedQuery(name = "Customer.findByAddressline2", query = "SELECT c FROM Customer c WHERE c.addressline2 = :addressline2"),
32     @NamedQuery(name = "Customer.findByCity", query = "SELECT c FROM Customer c WHERE c.city = :city"),
33     @NamedQuery(name = "Customer.findByState", query = "SELECT c FROM Customer c WHERE c.state = :state")
34 })

```

Choose getResultList().



And then the get the first element in the results:

```

PrintWriter out = response.getWriter();
try {
    // TODO output your page here
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet TestServlet</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Servlet TestServlet at " + request.getContextPath () + "</h1>");
    out.println("<h1>" + bean.SendHello("The Boss") + "</h1>");
    Customer cust = emf.createEntityManager().createNamedQuery("Customer.findAll").getResultList().get(0);
    out.println("</body>");
    out.println("</html>");

} finally {
    out.close();
}
}

```

Parameters:

index - index of the element to return

Returns:

the element at the specified position in this list

Throws:

[IndexOutOfBoundsException](#) - if the index is out of range (index < 0 || index >= size())

HttpServlet methods. Click on the + sign on the left to edit the code.

- add(Object e)
- add(int index, Object element)
- addAll(Collection c)
- addAll(int index, Collection c)
- clear()
- contains(Object o)
- containsAll(Collection c)
- equals(Object o)
- get(int index)**
- getClass()
- hashCode()
- indexOf(Object o)
- isEmpty()
- iterator()
- lastIndexOf(Object o)
- listIterator() List
- listIterator(int index) List

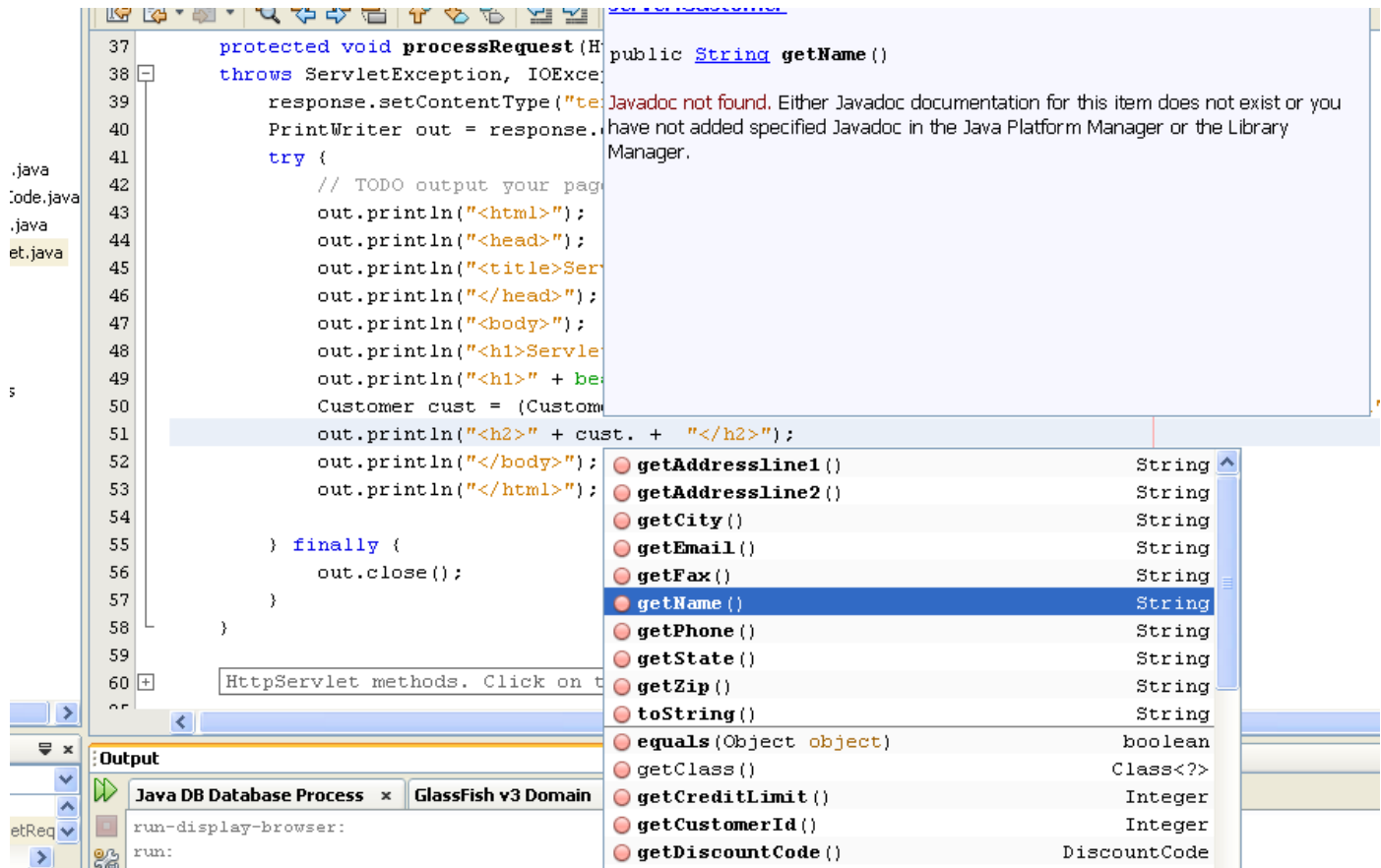
The full line code is:

```

out.println("<h1>Servlet TestServlet at " + request.getContextPath () + "</h1>");
out.println("<h1>" + bean.SendHello("The Boss") + "</h1>");
Customer cust = (Customer)emf.createEntityManager().createNamedQuery("Customer.findAll").getResultList().get(0);
out.println("</body>");
out.println("</html>");

```

Now we need to print the results in the web page:



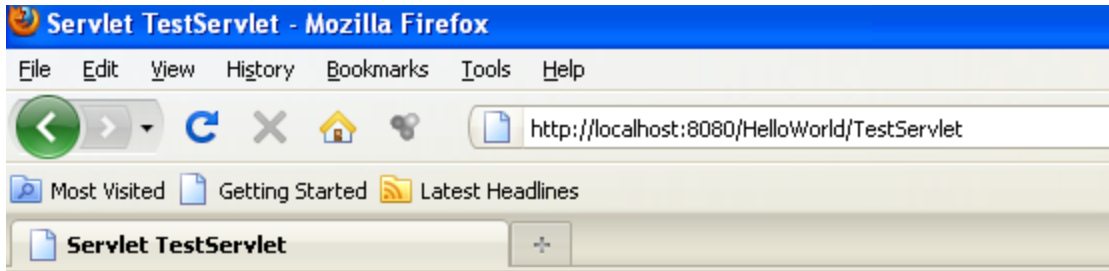
The line of printing is:

```

out.println("<h1>Servlet TestServlet at " + request.
out.println("<h1>" + bean.SendHello("The Boss") + "
Customer cust = (Customer) emf.createEntityManager().
out.println("<h2>" + cust.getName() + "</h2>");
out.println("</body>");
out.println("</html>");

```

Now if you run the application:



Servlet TestServlet at /HelloWorld

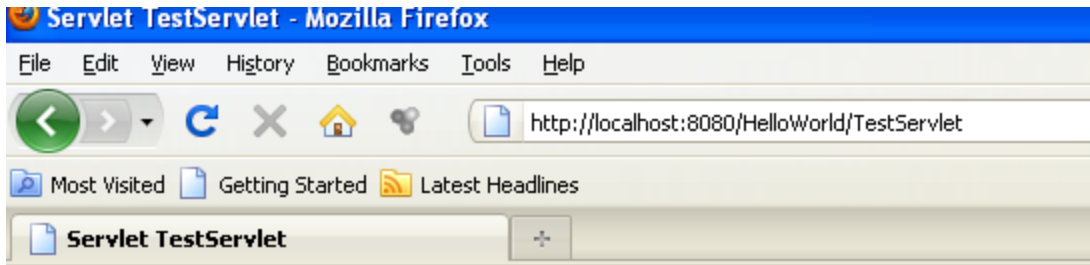
Name: The Boss

JumboCom

If you want to change the code in order to get the city of the customer:

```
out.println("<body>");
out.println("<h1>Servlet TestServlet at " + request.get
out.println("<h1>" + bean.SendHello("The Boss") + "</h1>");
Customer cust = (Customer)emf.createEntityManager().create
out.println("<h2>" + cust.getName() + "</h2>");
out.println("<h2>" + cust.getCity() + "</h2>");
out.println("</body>");
out.println("</html>");
```

When you refresh the browser:



Servlet TestServlet at /HelloWorld

Name: The Boss

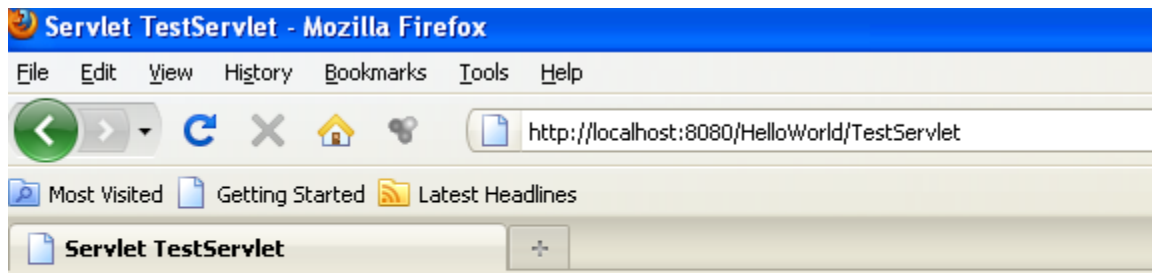
JumboCom

Fort Lauderdale

Some more changes:

```
out.println("<body>");
out.println("<h1>Servlet TestServlet at " + request.getContextPath () + "</h1>");
out.println("<h1>" + bean.SendHello("The Boss") + "</h1>");
Customer cust = (Customer)emf.createEntityManager().createNamedQuery("Customer.");
out.println("<h2> The name of the customer is: " + cust.getName() + "</h2>");
out.println("<h2> The city of the customer is: " + cust.getCity() + "</h2>");
out.println("</body>");
out.println("</html>");
```

When you refresh the browser:



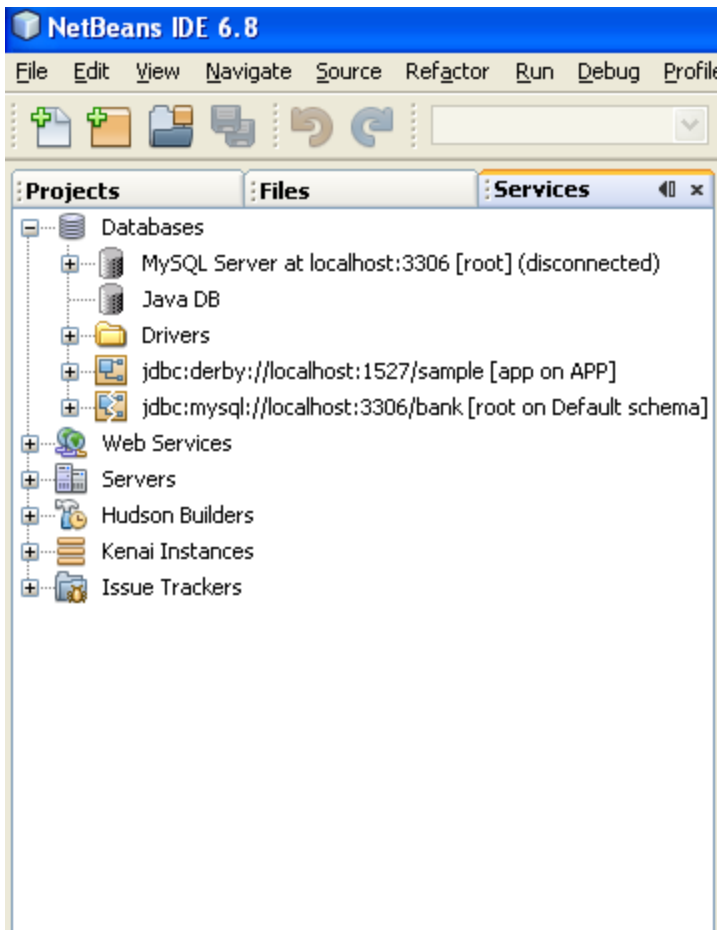
Servlet TestServlet at /HelloWorld

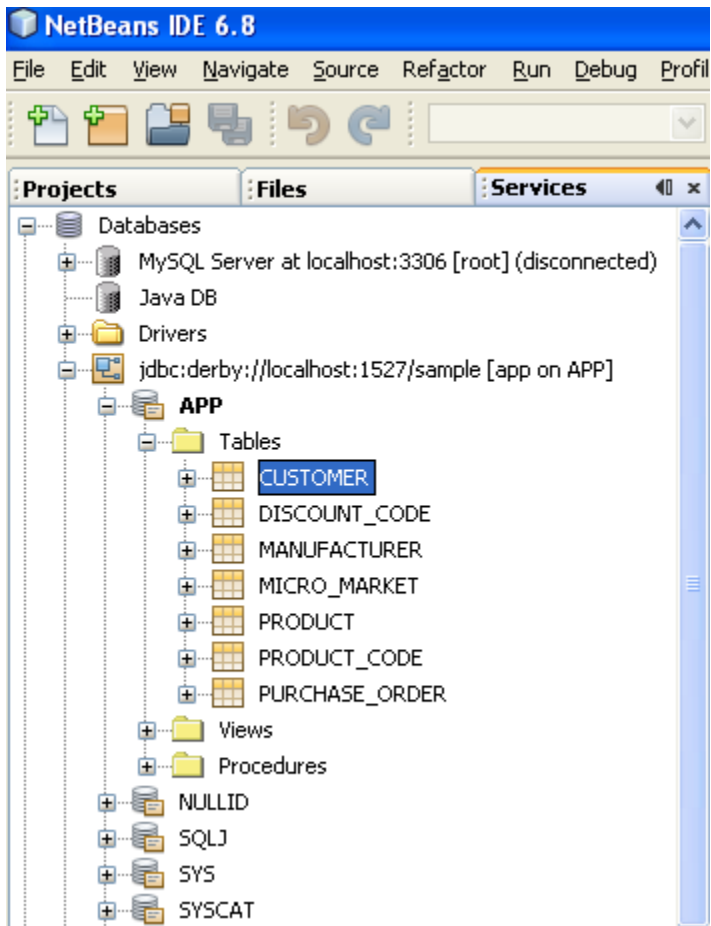
Name: The Boss

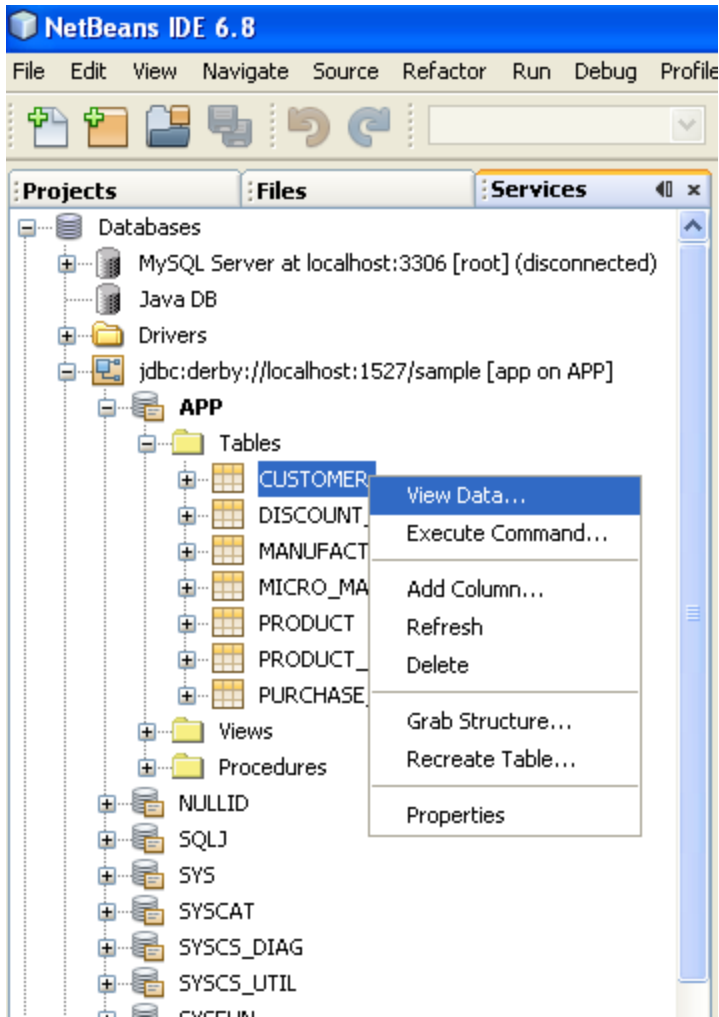
The name of the customer is: JumboCom

The city of the customer is: Fort Lauderdale

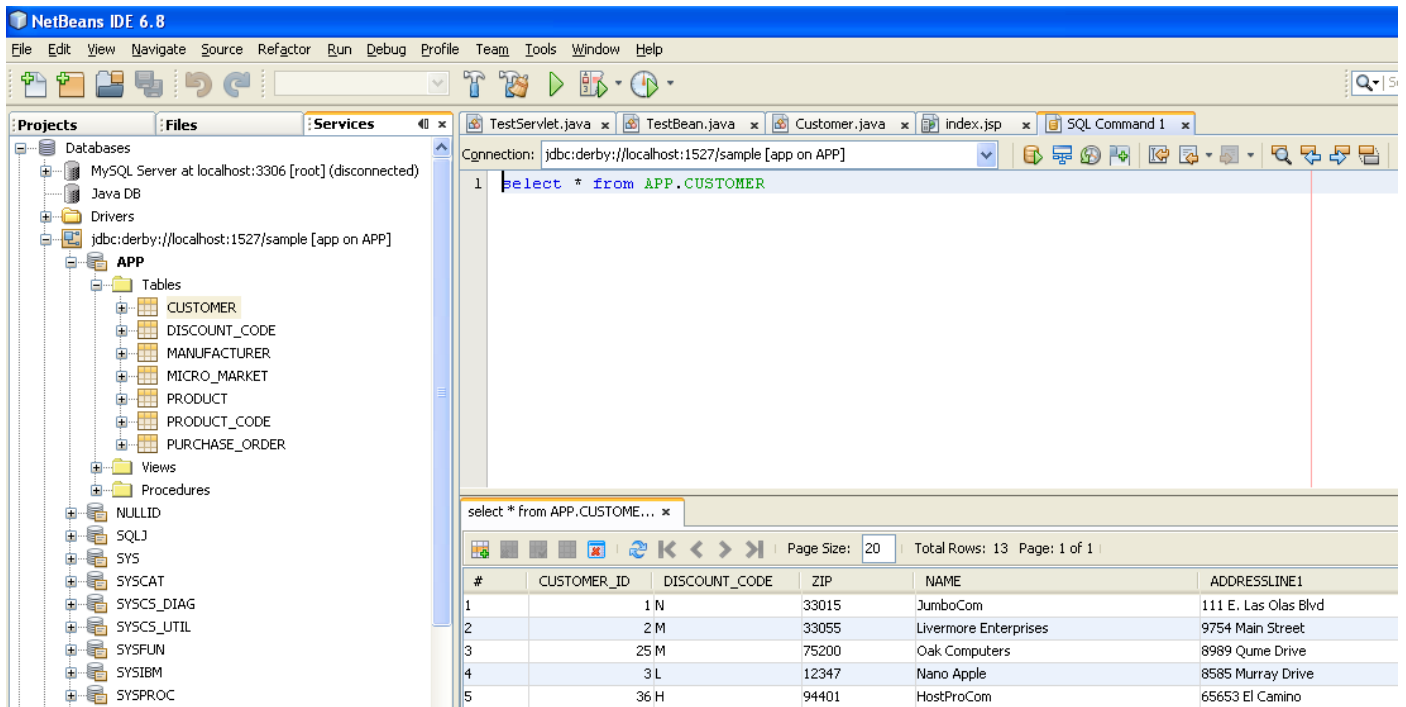
The database to which we are connected can be seen at:







You will see the following editor of SQL and the data of the table:



5. Developing a web banking application

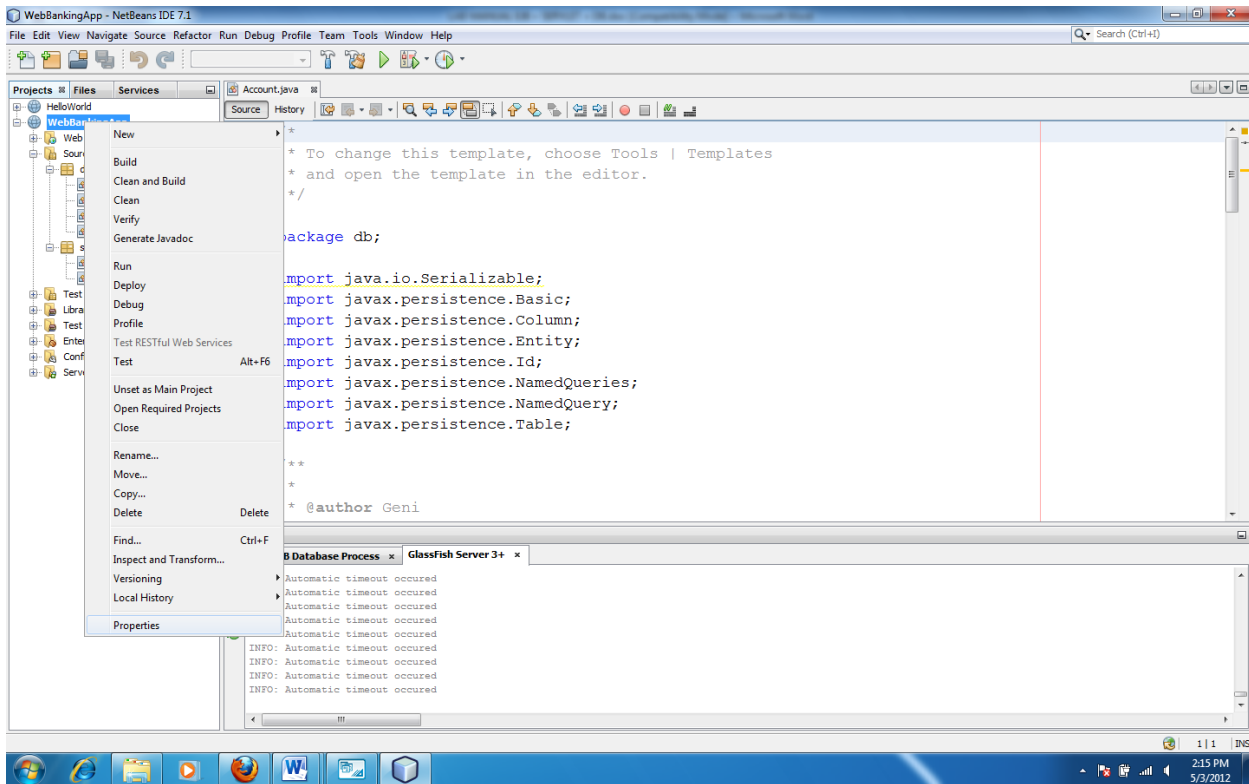
We will develop a web banking application that connects to the MySQL database.

Perform the following in order:

In order to connect Java with MySQL we need the following connector:

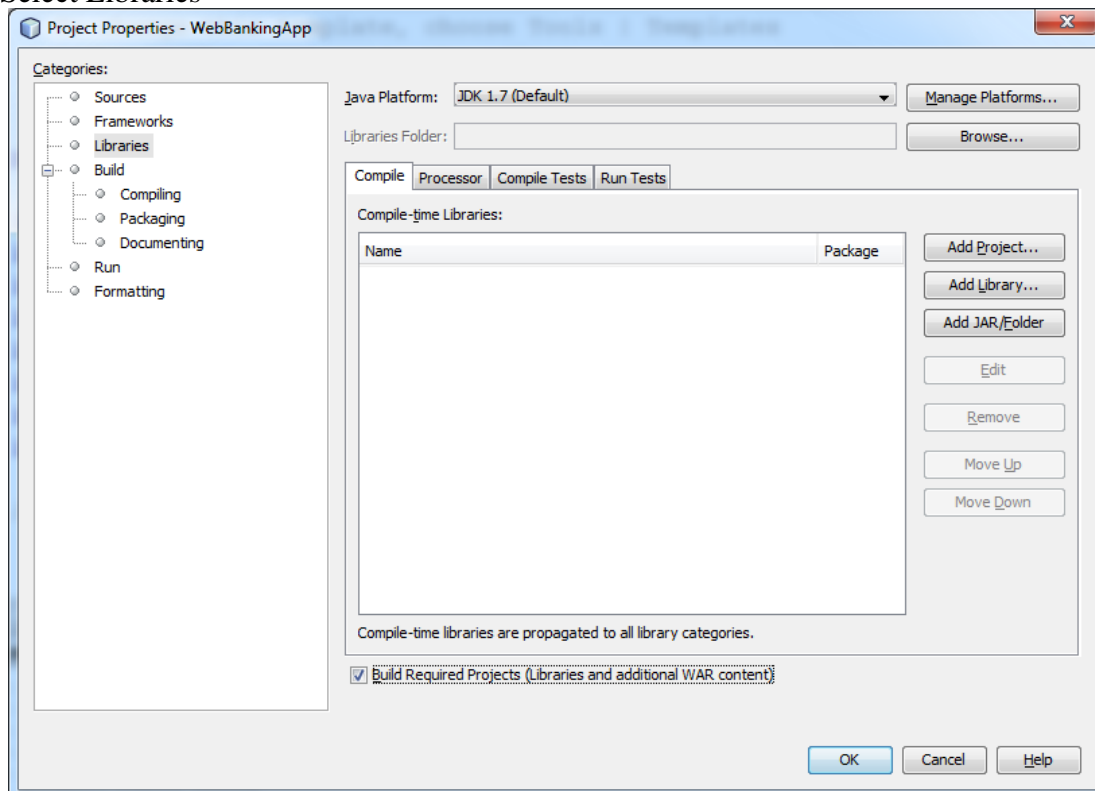
mysql-connector-java-5.1.15-bin.jar

Add the connector to the libraries of the project in Netbeans as follows:

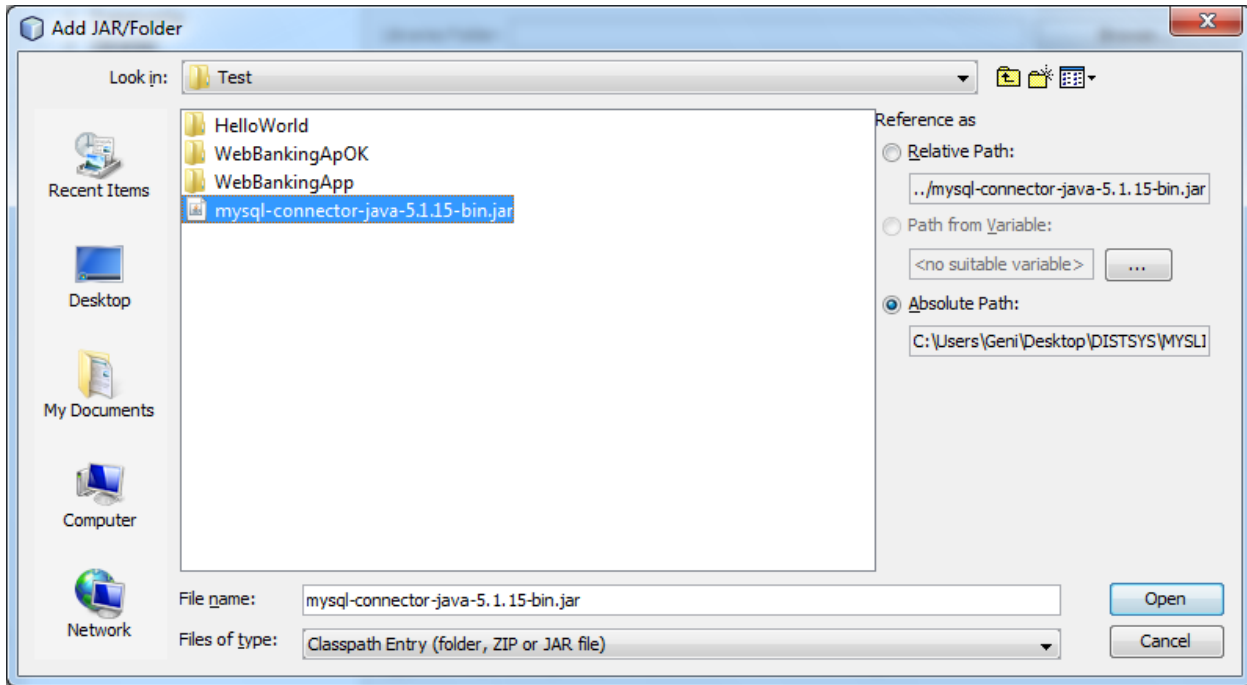


Click with right of the mouse on the Project. Select Properties

Select Libraries



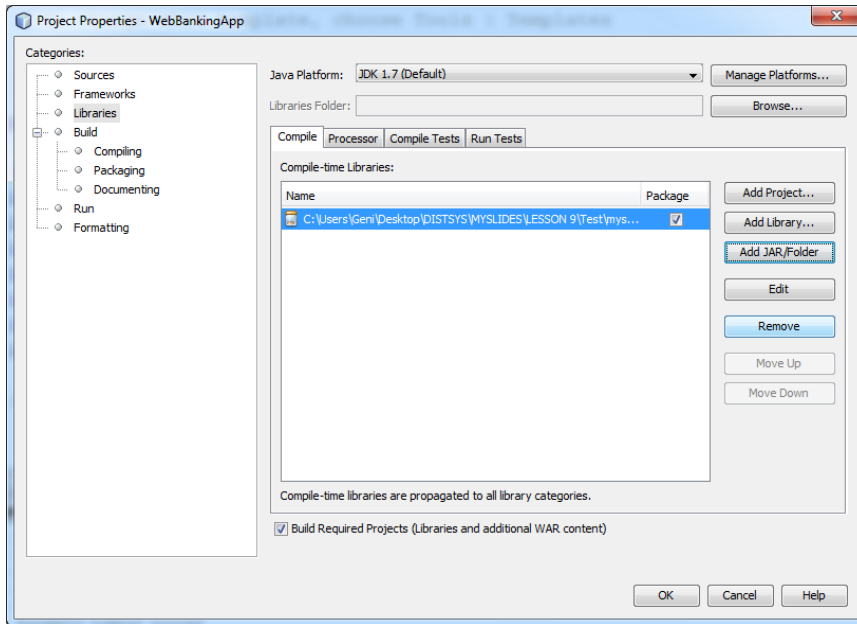
Select on the right “Add JAR/Folder”:



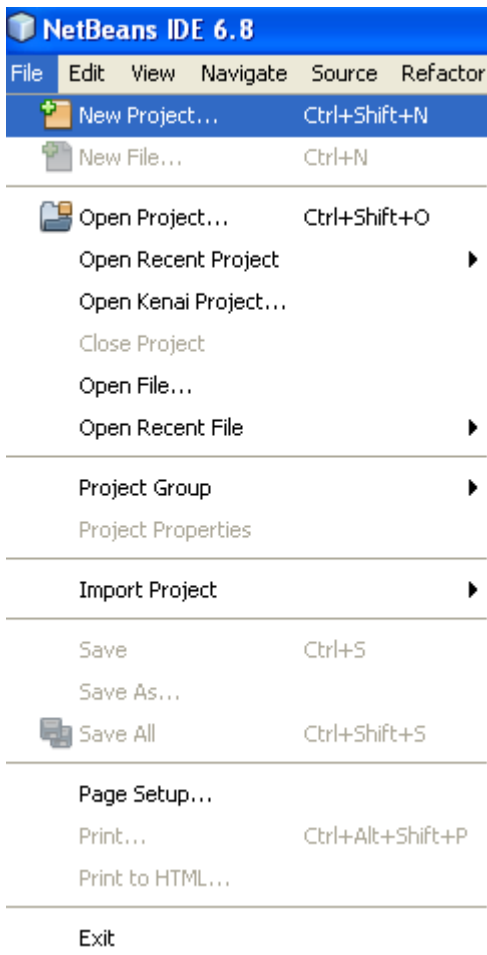
Select the file

mysql-connector-java-5.1.15-bin.jar

Now the Java program is properly set to connect to MySQL.



Create the project:



New Project



Steps

1. **Choose Project**
2. ...

Choose Project

Categories:

- Java
- JavaFX
- Java Web**
- Java EE
- Java ME
- Maven
- NetBeans Modules
- +
- Samples

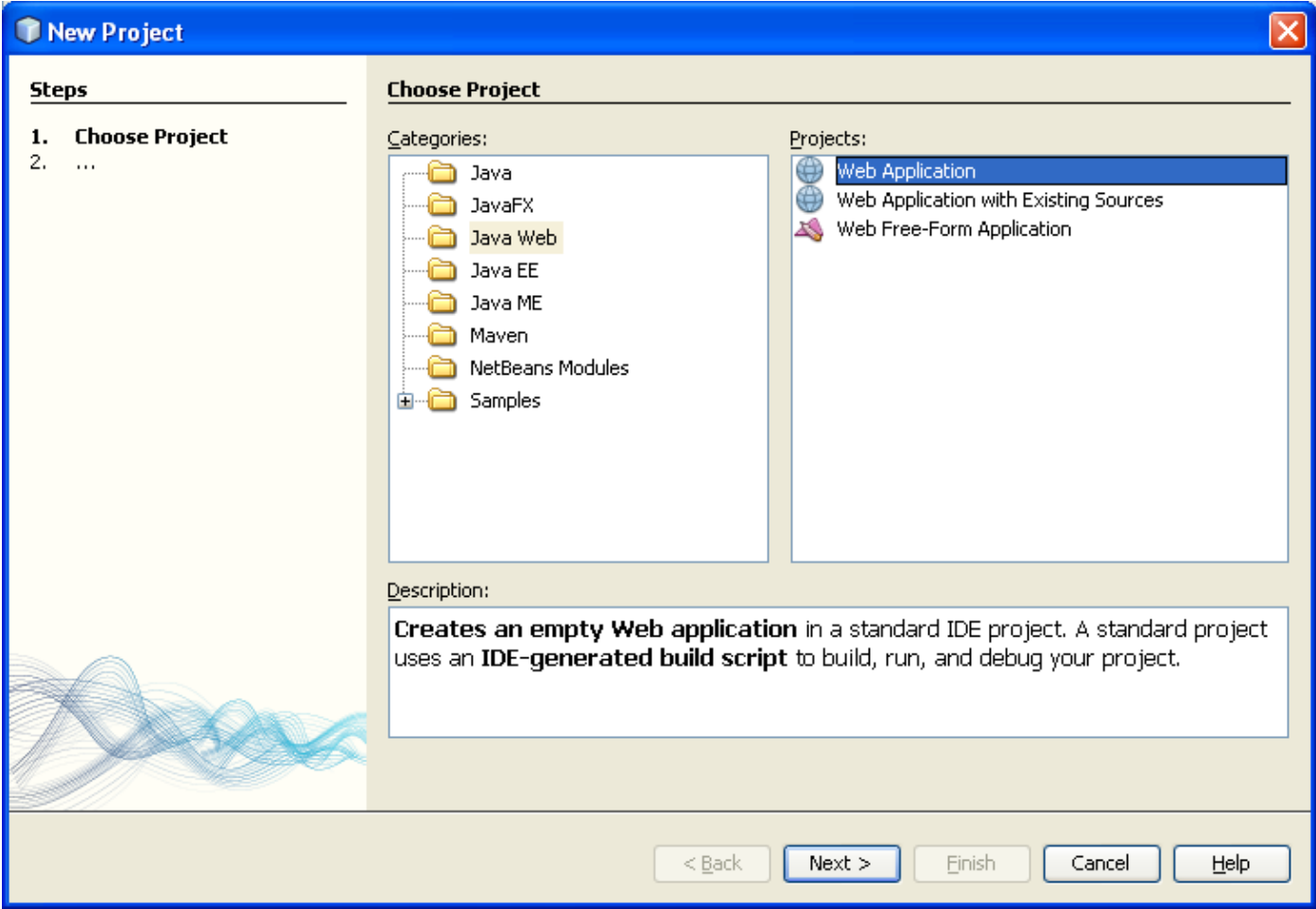
Projects:

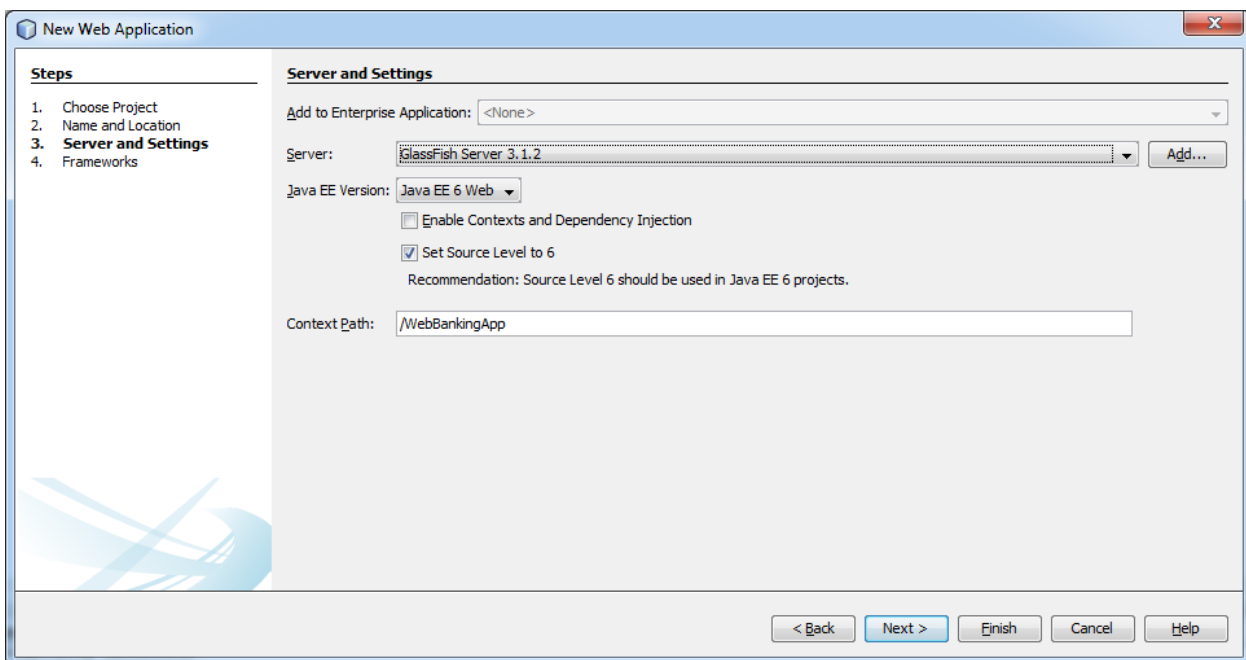
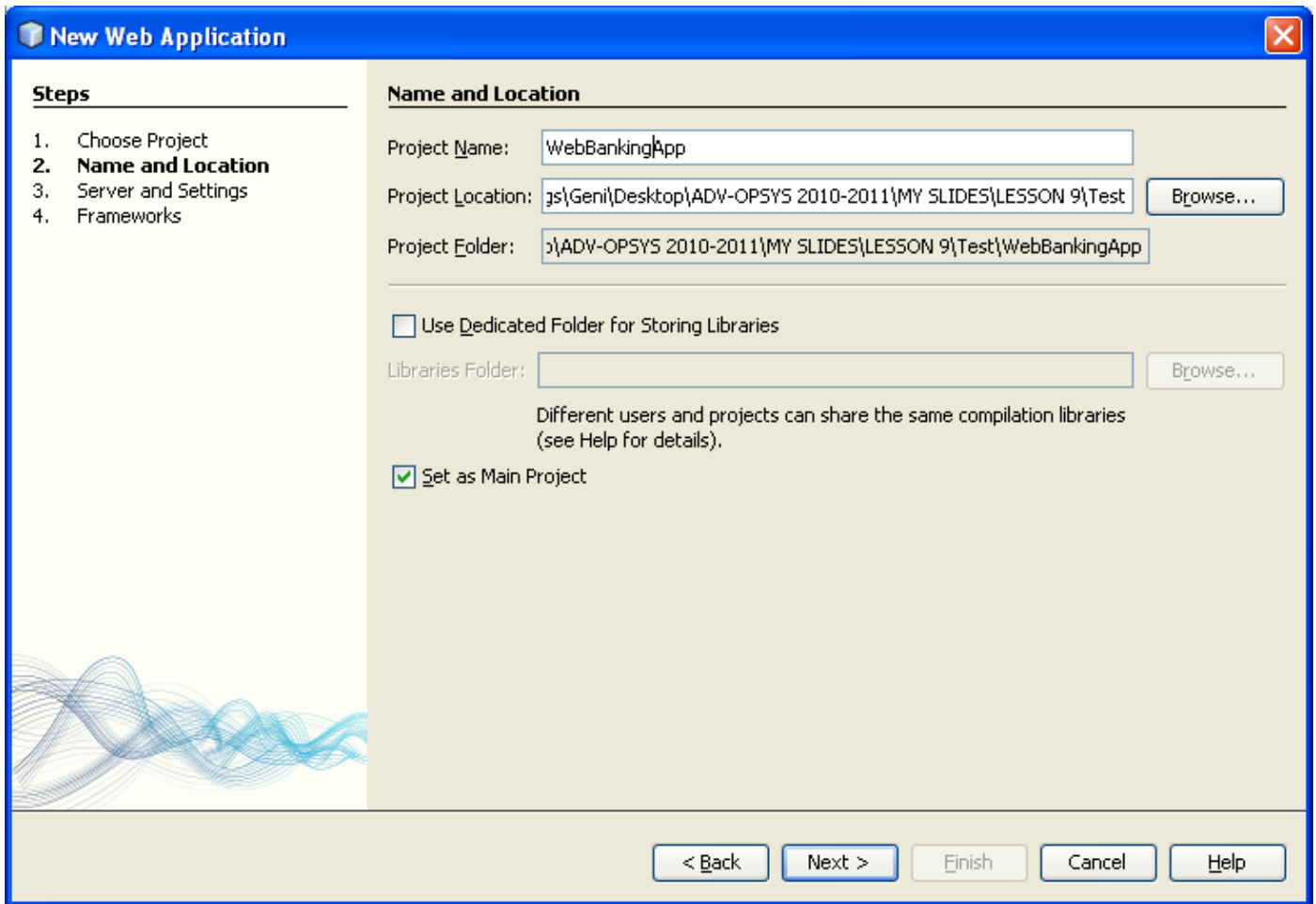
- Web Application
- Web Application with Existing Sources
- Web Free-Form Application

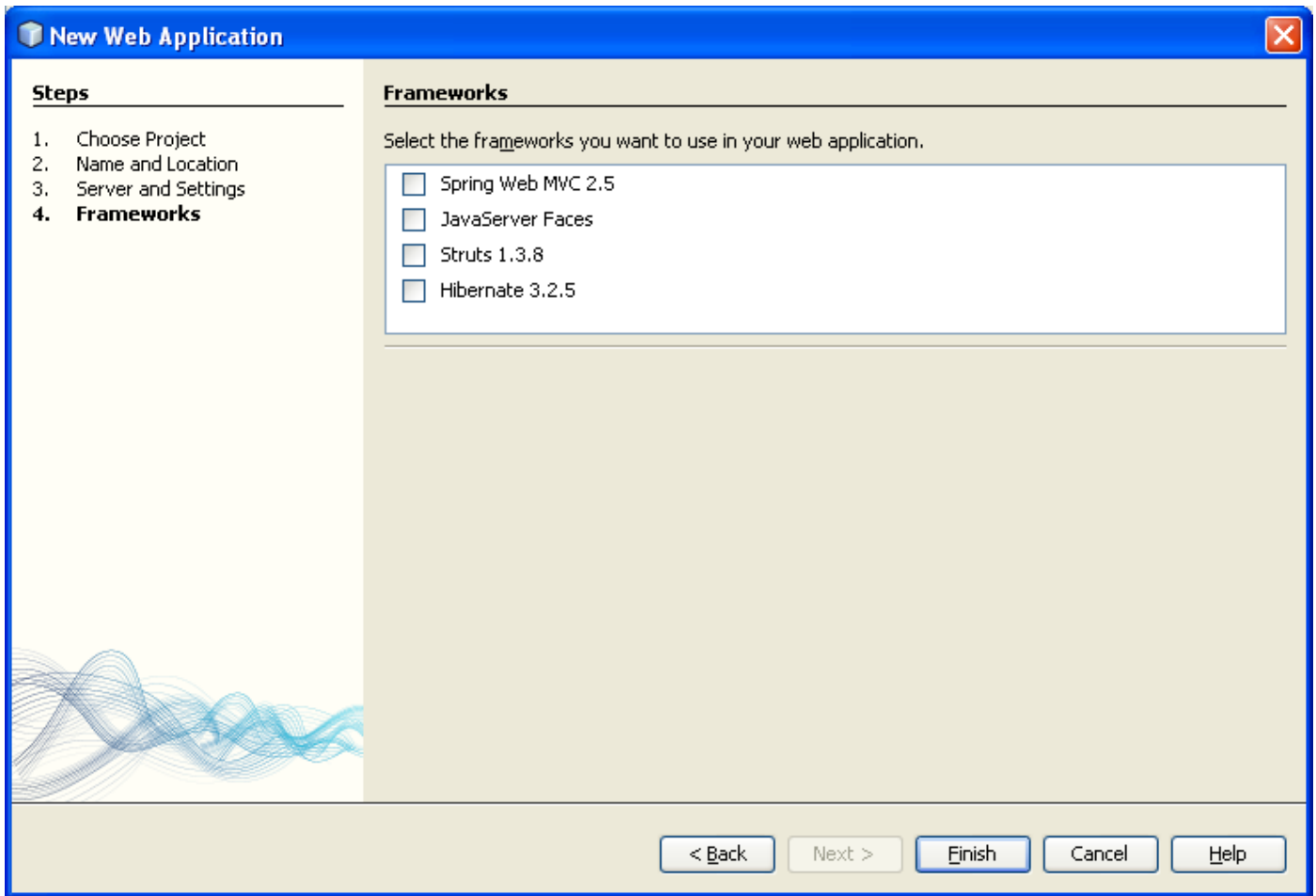
Description:

Creates an empty Web application in a standard IDE project. A standard project uses an **IDE-generated build script** to build, run, and debug your project.

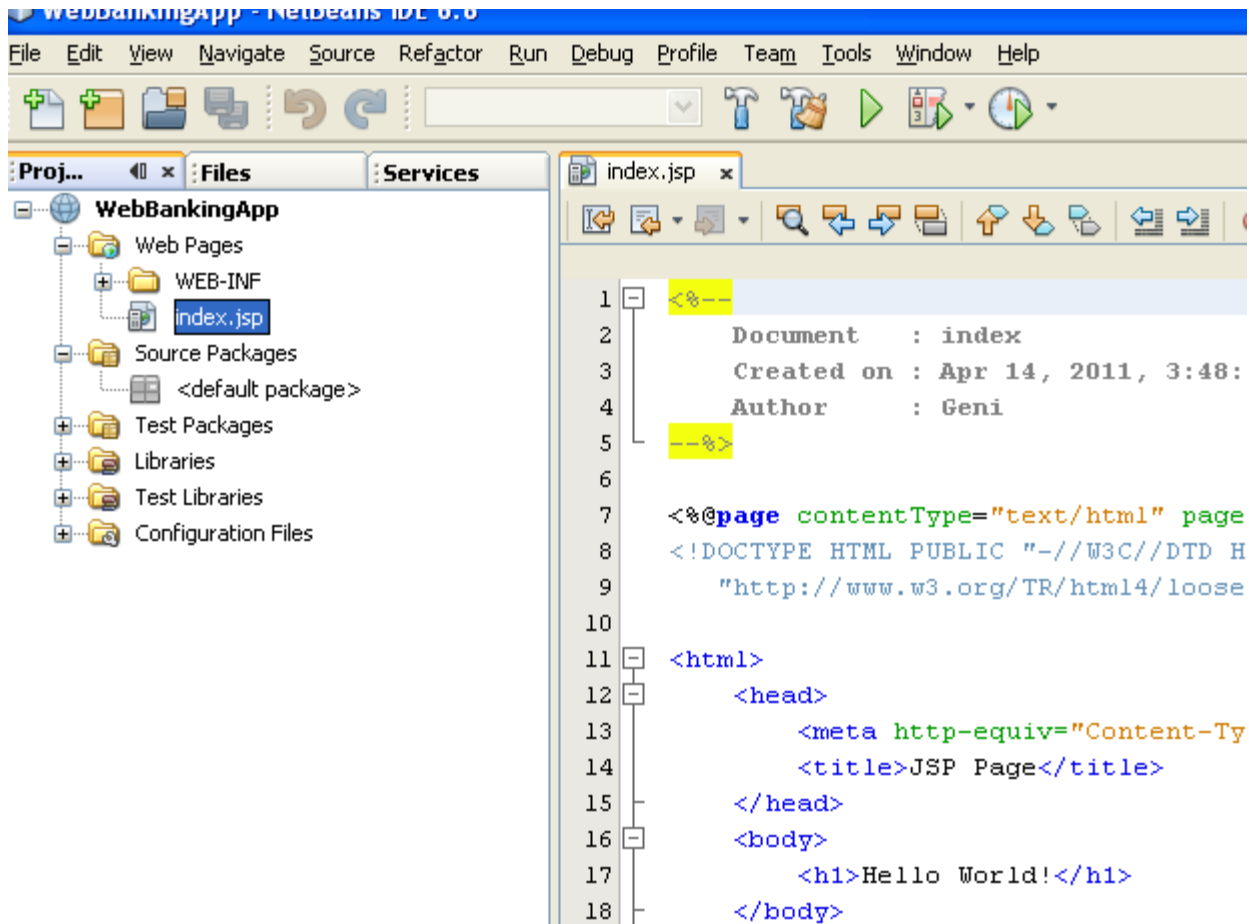
- < Back **Next >** Finish Cancel Help



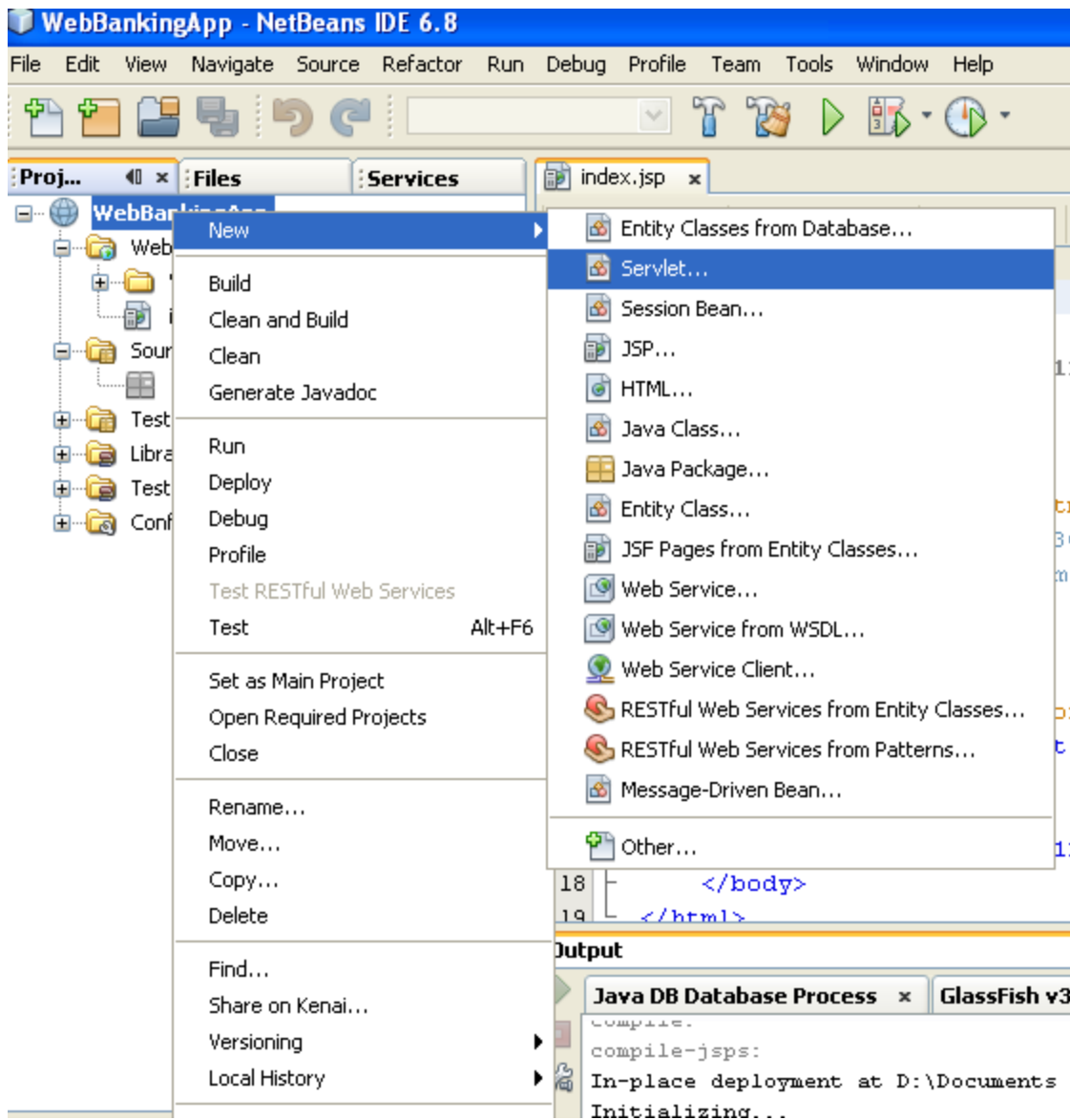




The application created is this one:



Create a Servlet:



New Servlet [Close]

Steps

1. Choose File Type
- 2. Name and Location**
3. Configure Servlet Deployment

Name and Location

Class Name:


Project:

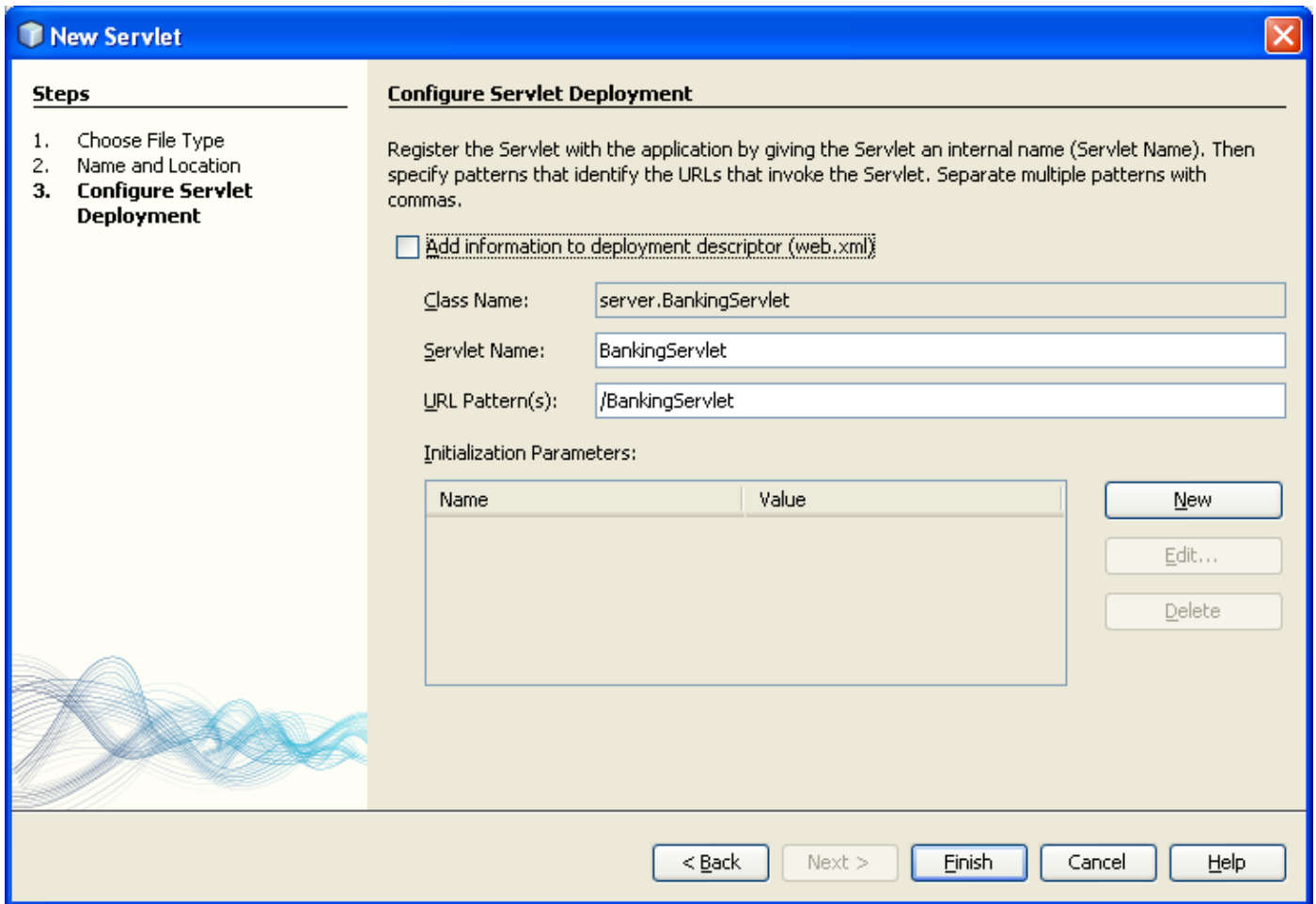
Location: ▼

Package: ▼

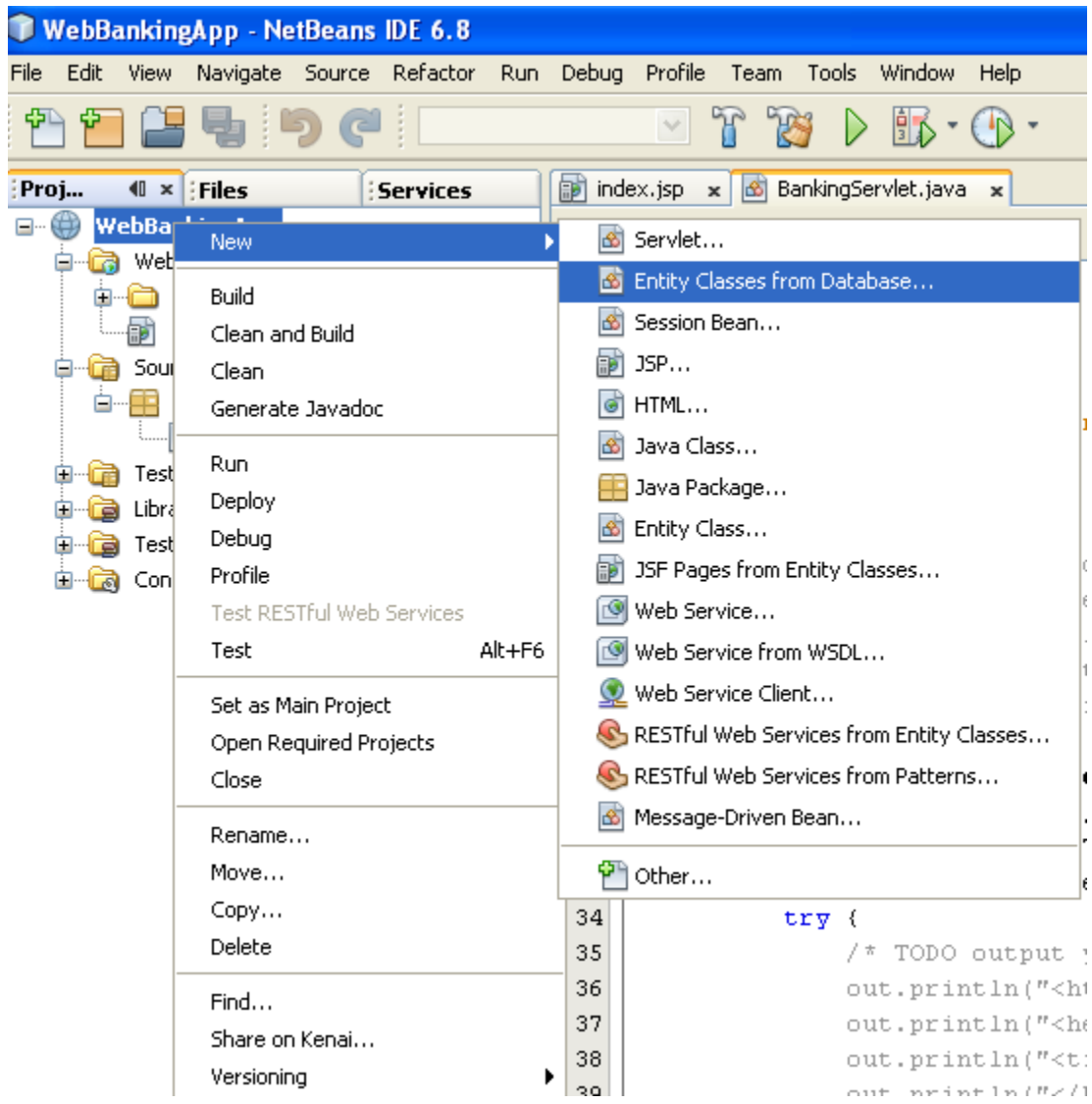
Created File:

< Back Next > Finish Cancel Help





Create an EJB to connect to the database MySQL:



New Entity Classes from Database

Steps

1. Choose File Type
- 2. Database Tables**
3. Entity Classes
4. Mapping Options

Database Tables

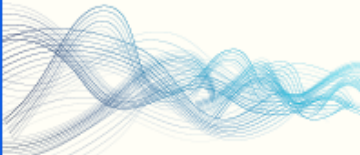
Data Source:


Database Schema: <no database schemas in the project>

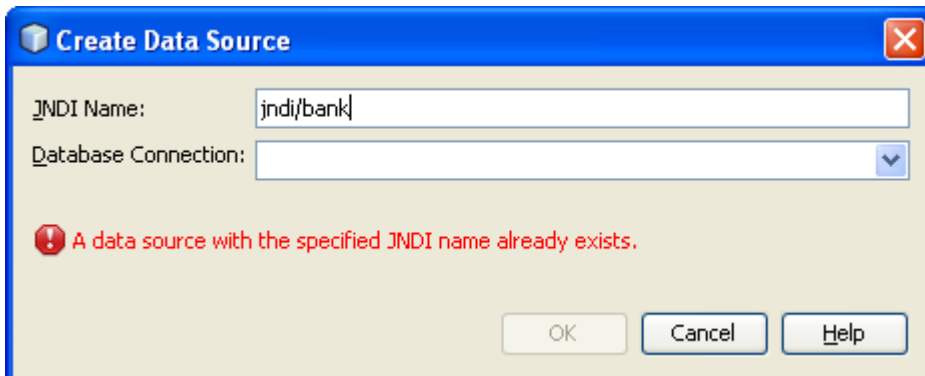
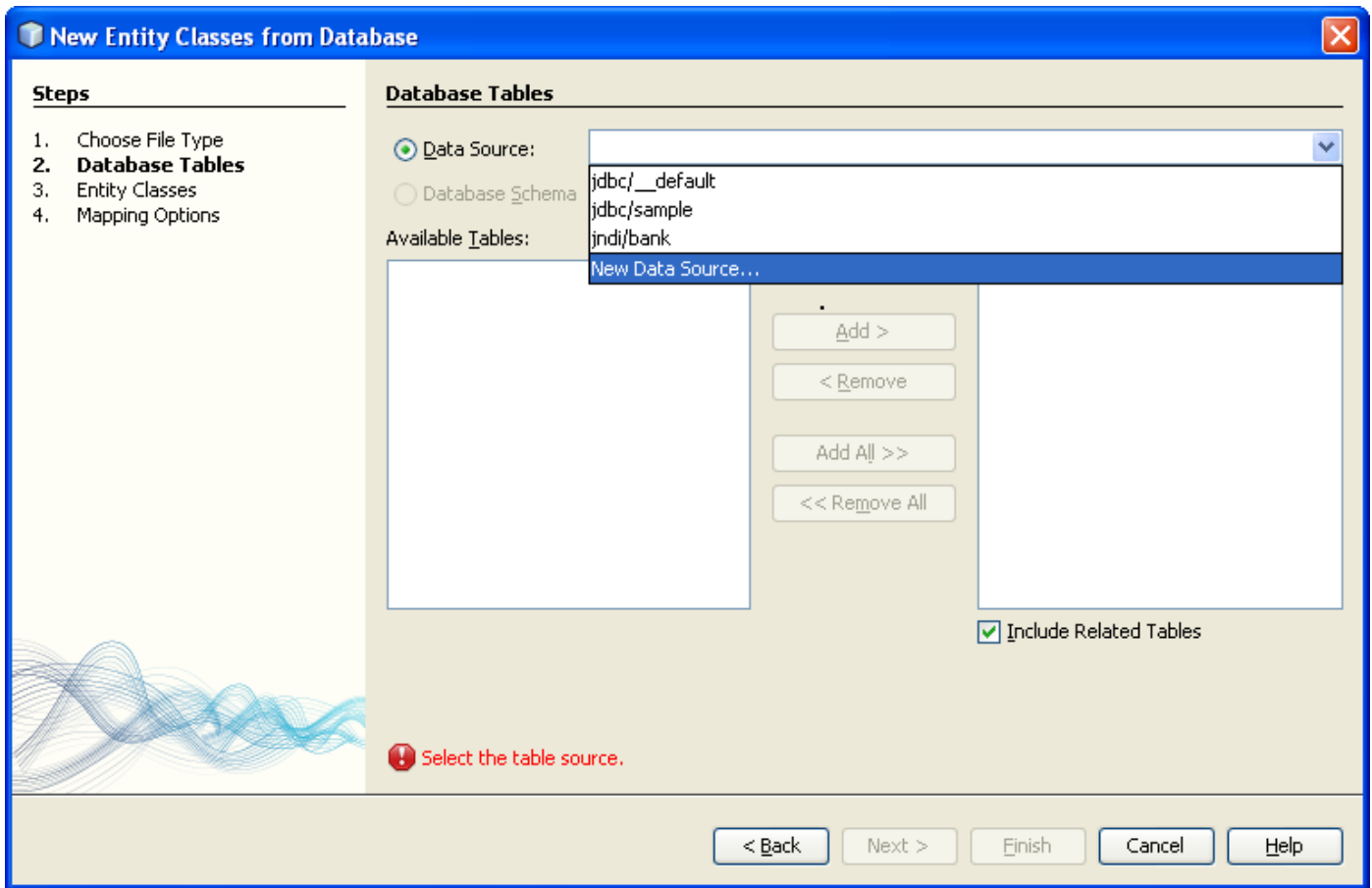
Available Tables:

Selected Tables:

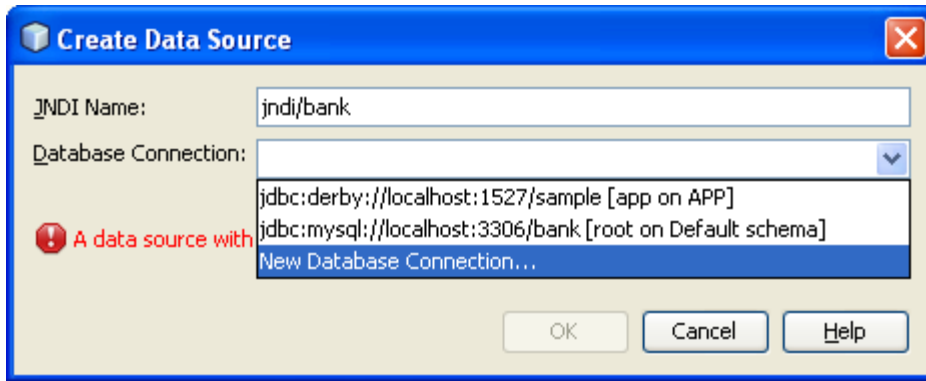
Include Related Tables



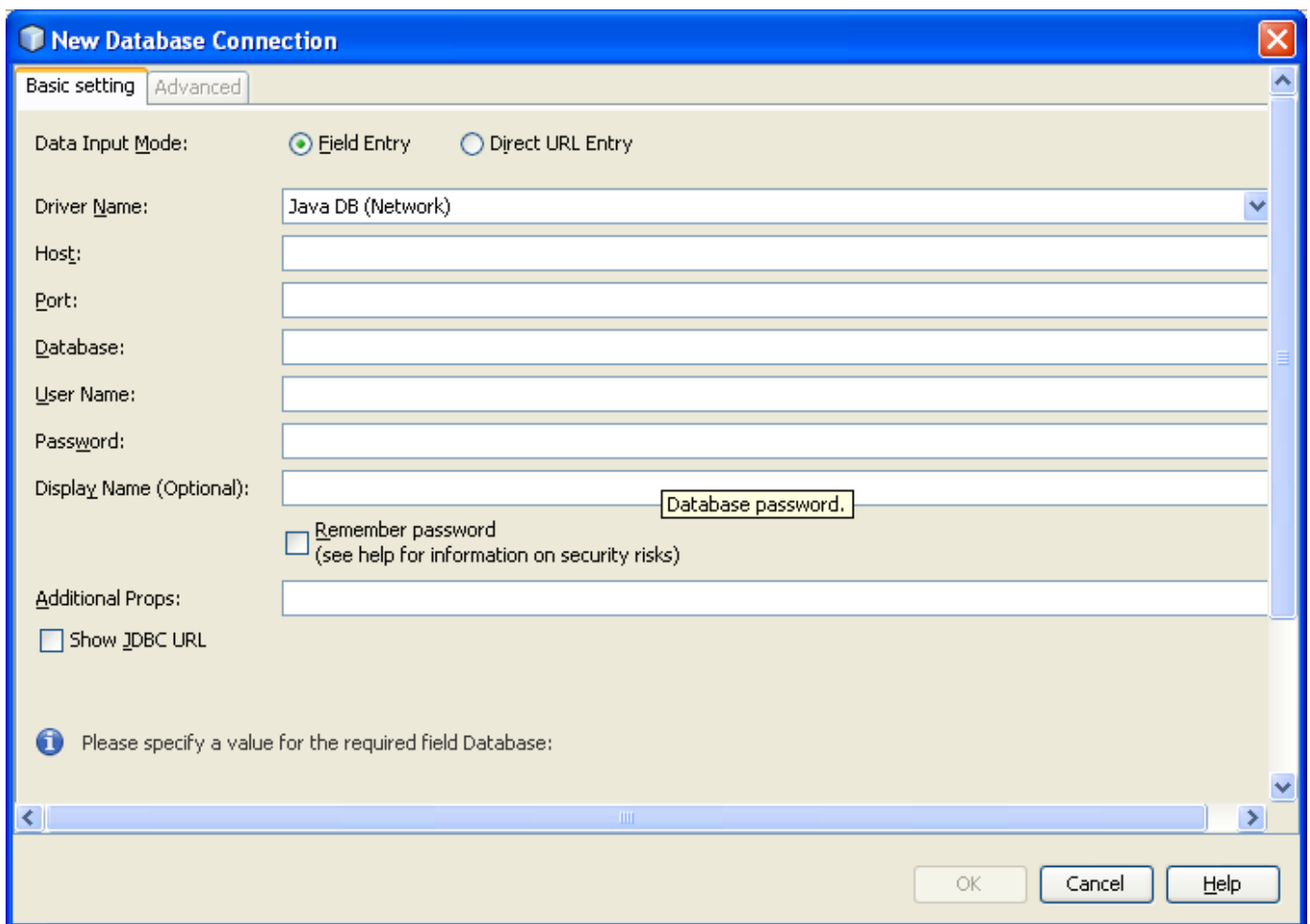
 Select the table source.

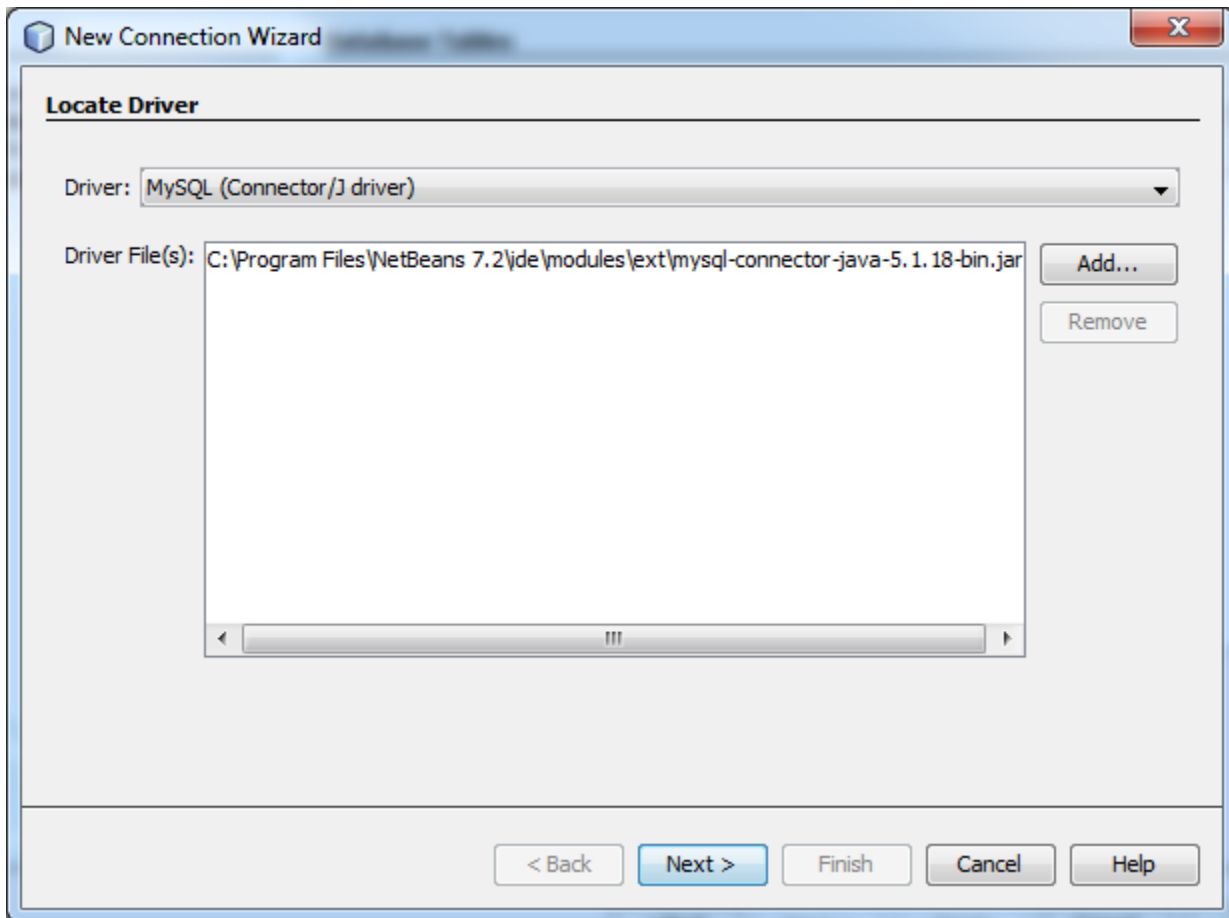


If you do not have the connection to MySQL select “New Database Connection” as follows:



The following page appears:





Fill in the fields as follows;

New Connection Wizard

Customize Connection

Driver Name: MySQL (Connector/J driver)

Host: localhost Port: 3306

Database: bank

User Name: root

Password: ●●●●

Remember password

Test Connection

JDBC URL: jdbc:mysql://localhost:3306/bank?zeroDateTimeBehavior=convertToNull

< Back Next > Finish Cancel Help

Press Test Connection to check your connection:

New Connection Wizard

Customize Connection

Driver Name: MySQL (Connector/J driver)

Host: localhost Port: 3306

Database: bank

User Name: root

Password: ●●●●

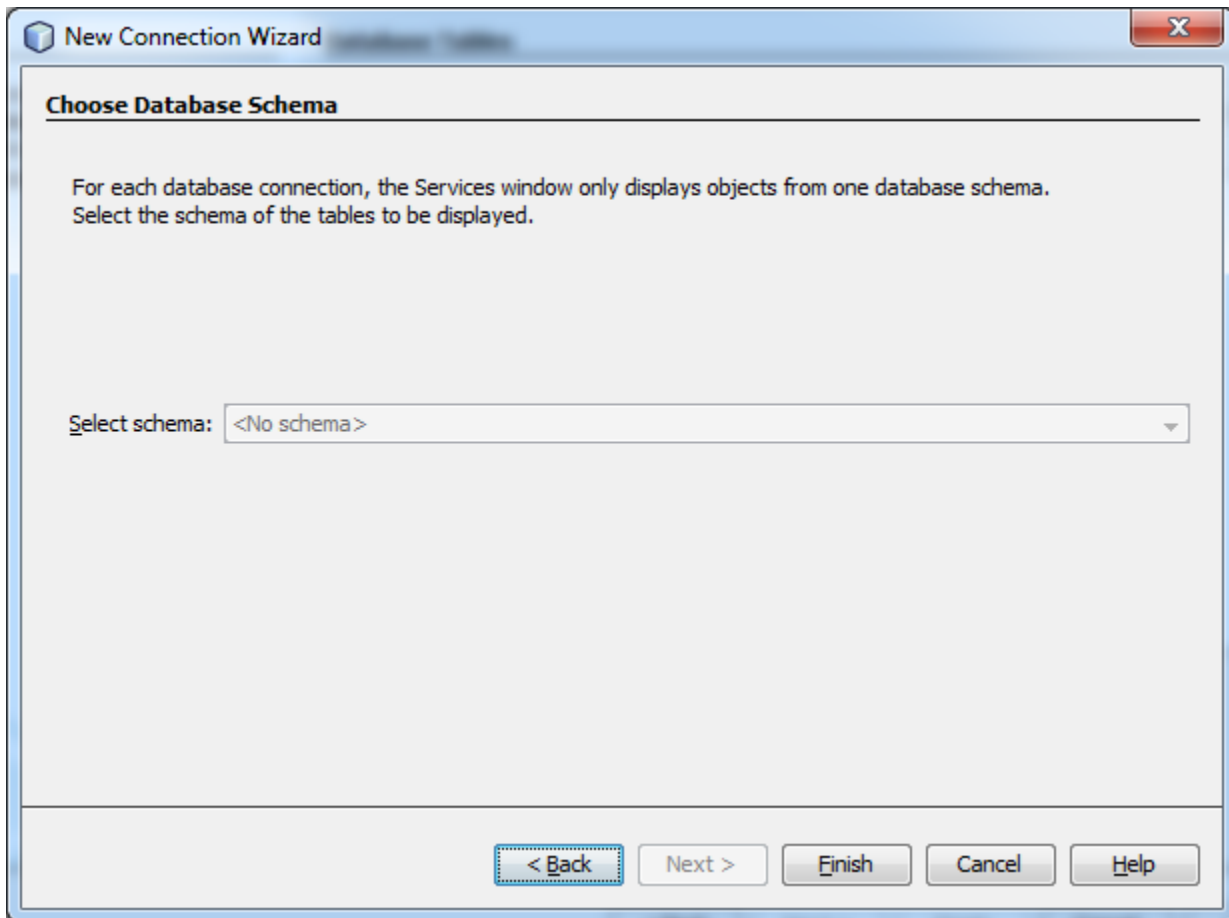
Remember password

Test Connection

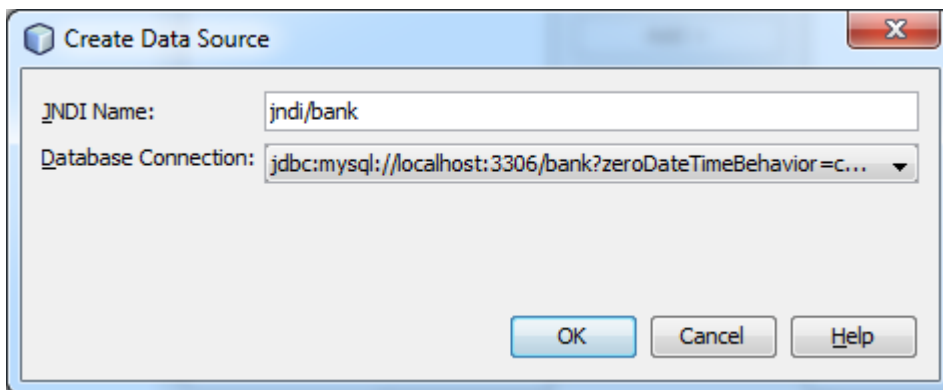
JDBC URL: jdbc:mysql://localhost:3306/bank?zeroDateTimeBehavior=convertToNull

i Connection Succeeded.

< Back Next > Finish Cancel Help

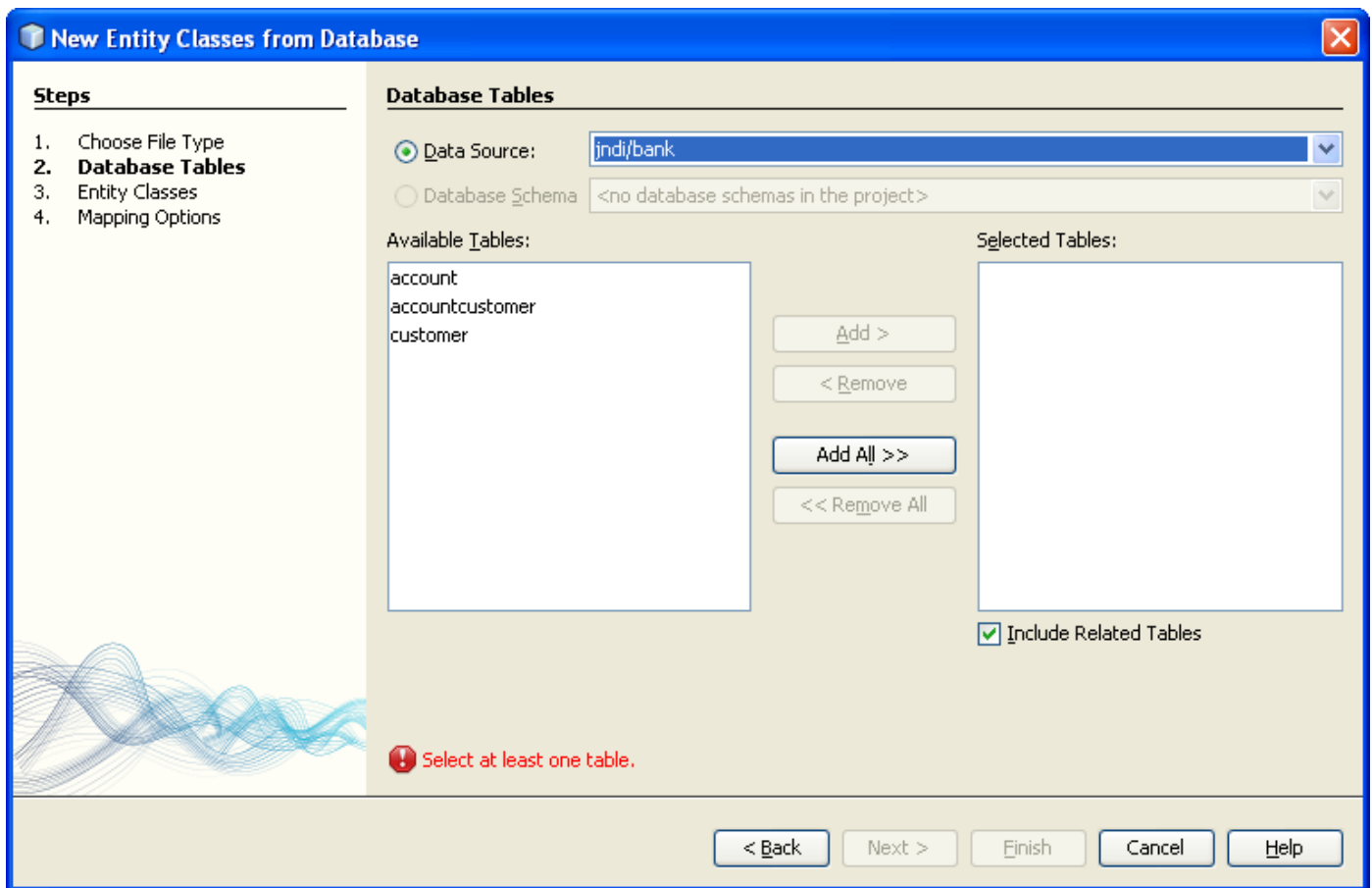


Press Finish:

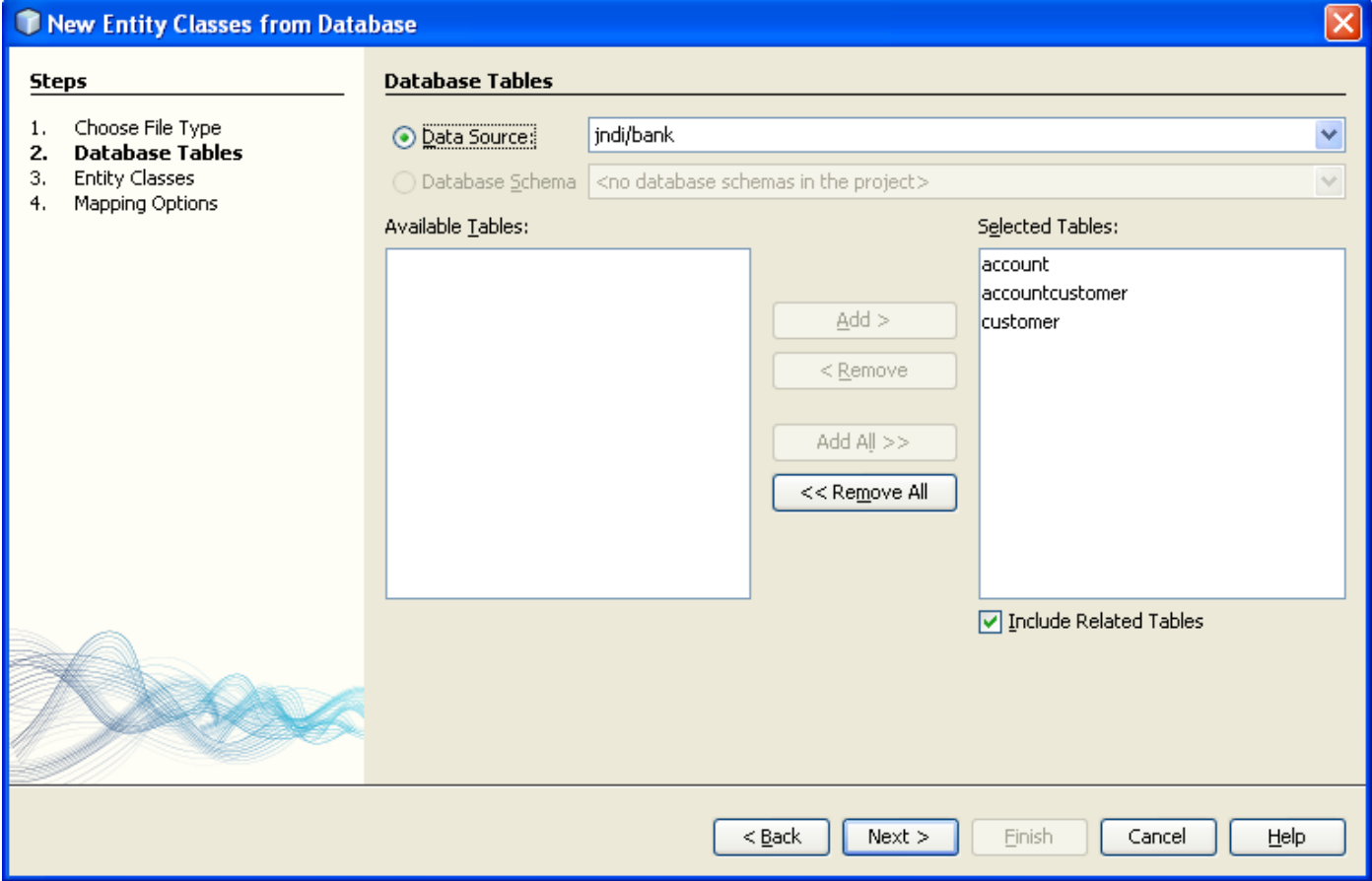


Press OK and you should have the following page:

Choose



Now select the tables with “Add All”:



New Entity Classes from Database

Steps

1. Choose File Type
2. Database Tables
- 3. Entity Classes**
4. Mapping Options

Entity Classes

Specify the names and the location of the entity classes.

Class Names:

Database Table	Class Name	Generation Type
account	Account	New
accountcustomer	Accountcustomer	New
customer	Customer	New

...

Project: WebBankingApp

Location: Source Packages

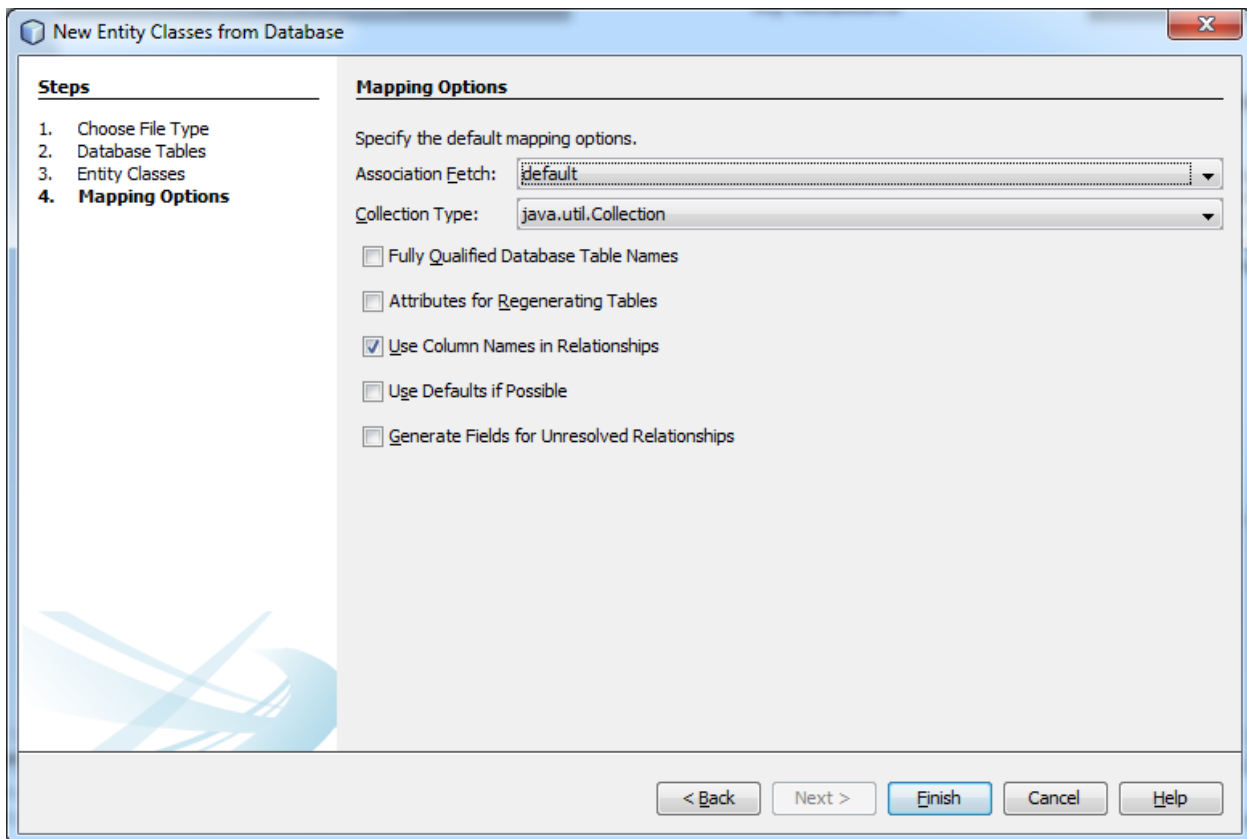
Package: db

Generate Named Query Annotations for Persistent Fields

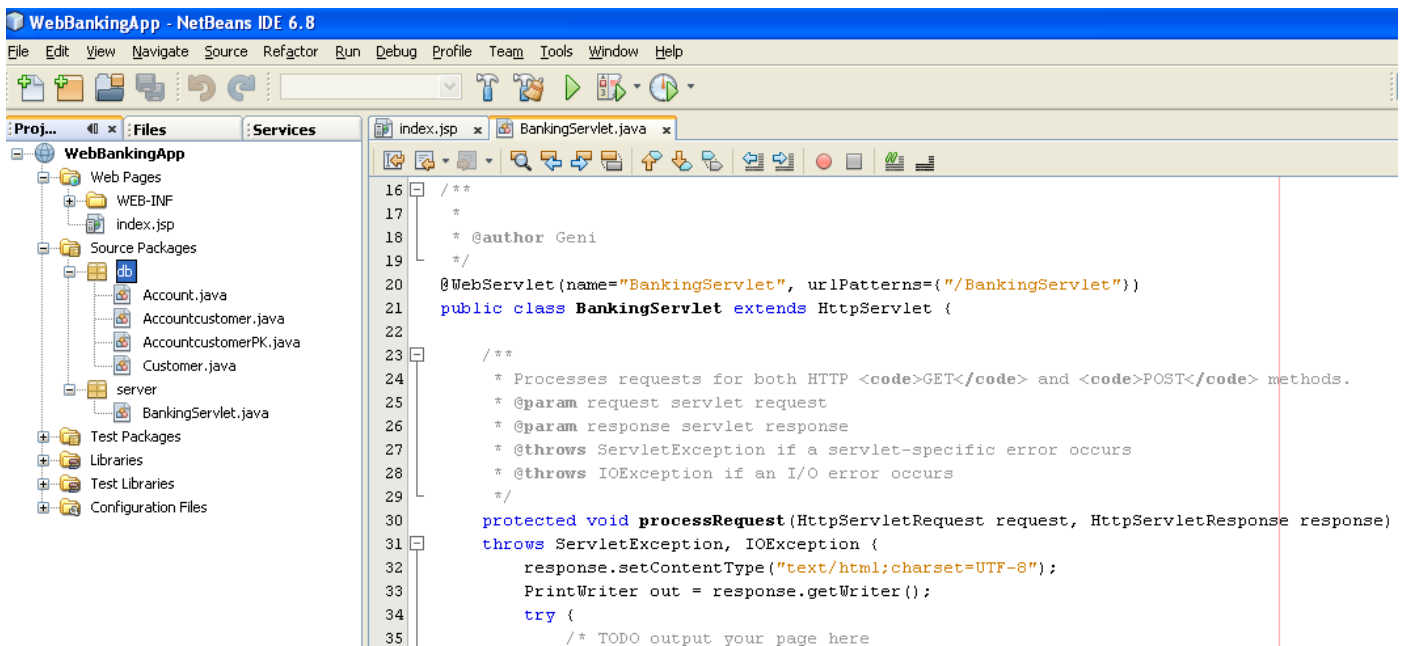
Generate JAXB Annotations

Create Persistence Unit

< Back Next > Finish Cancel Help



The following will be generated:



In BankingServlet code the following:

```

16  /**
17   *
18   * @author Geni
19   */
20  @WebServlet(name="BankingServlet", urlPatterns={"/BankingServlet"})
21  public class BankingServlet extends HttpServlet {
22
23      @PersistenceUnit
24      EntityManagerFactory emf;
25
26      /**
27       * Processes requests for both HTTP <code>GET</code> and <code>POST</code>
28       * @param request servlet request

```

Arrange the imports. Click on the red point on the left shown by NetBeans.

```

15
16  /**
17   *
18   * @author Geni
19   */
20  @WebServlet(name="BankingServlet", urlPatterns={"/BankingServlet"})
21  public class BankingServlet extends HttpServlet {
22
23      @PersistenceUnit
24
25

```

cannot find symbol
symbol: class PersistenceUnit
location: class server.BankingServlet

Add import for javax.persistence.PersistenceUnit

```

20  /**
21  @WebServlet(name="BankingServlet", urlPatterns={"/BankingServlet"})
22  public class BankingServlet extends HttpServlet {
23
24      EntityManagerFactory emf;
25
26      /**
27       * Processes requests for both HTTP <code>GET</code> and <code>POST</code>
28       * @param request servlet request

```

cannot find symbol
symbol: class EntityManagerFactory
location: class server.BankingServlet

Add import for javax.persistence.EntityManagerFactory
Create class "EntityManagerFactory" in package server

You will see that in the imports there are two more lines:

```

8 import java.io.IOException;
9 import java.io.PrintWriter;
10 import javax.persistence.EntityManagerFactory;
11 import javax.persistence.PersistenceUnit;
12 import javax.servlet.ServletException;
13 import javax.servlet.annotation.WebServlet;
14 import javax.servlet.http.HttpServlet;
15 import javax.servlet.http.HttpServletRequest;
16 import javax.servlet.http.HttpServletResponse;
17

```

Now we go the point of generating the webpage.

```

38     PrintWriter out = response.getWriter();
39     try {
40         /* TODO output your page here
41         out.println("<html>");
42         out.println("<head>");
43         out.println("<title>Servlet BankingServlet</title>");
44         out.println("</head>");
45         out.println("<body>");
46         out.println("<h1>Servlet BankingServlet at " + request.getContextPath () + "</h1>");
47         out.println("</body>");
48         out.println("</html>");
49         */
50     } finally {
51         out.close();
52     }
53 }

```

Uncomment the commented part as follows:

```

38     PrintWriter out = response.getWriter();
39     try {
40         // TODO output your page here
41         out.println("<html>");
42         out.println("<head>");
43         out.println("<title>Servlet BankingServlet</title>");
44         out.println("</head>");
45         out.println("<body>");
46         out.println("<h1>Servlet BankingServlet at " + request.getContextPath () + "</h1>");
47         out.println("</body>");
48         out.println("</html>");
49
50     } finally {
51         out.close();
52     }
53 }

```

Now let us use the EJB we created. Perform the following operations:

The screenshot shows an IDE with the following components:

- Code Editor:** Displays `BankingServlet.java` with lines 36-57. The code includes a try-catch block for `EntityManagerFactory` and a `finally` block for `out.close()`.
- Documentation Window:** Shows the `EntityManagerFactory` class with the `createEntityManager()` method signature and description. It also lists various methods like `close()`, `getCache()`, and `isOpen()`.
- Output Console:** Shows the output of the application, including the text "HttpServlet methods. Click" and "initializing..."

Then:

The screenshot shows an IDE with the following components:

- BankingServlet.java (lines 36-57):**

```

36 throws ServletException, IOException {
37     response.setContentType("text/html;charset=UTF-8");
38     PrintWriter out = response.getWriter();
39     try {
40         // TODO output your page here
41         out.println("<html>");
42         out.println("<head>");
43         out.println("<title>Servlet BankingServlet");
44         out.println("</head>");
45         out.println("<body>");
46         out.println("<h1>Servlet BankingServlet");
47
48         Customer cust = emf.createEntityManager().
49
50         out.println("</body>");
51         out.println("</html>");
52
53     } finally {
54         out.close();
55     }
56 }
57

```
- Javadoc for `javax.persistence.EntityManager.createNamedQuery`:**

```

public Query createNamedQuery(String name)

Create an instance of Query for executing a named query (in the Java Persistence
query language or in native SQL).

Parameters:
    name - the name of a query defined in metadata

Returns:
    the new query instance

Throws:
    java.lang.IllegalArgumentException - if a query has not been defined
    with the given name or if the query string is found to be invalid

Method List:
- clear()
- close()
- contains(Object entity)
- createNamedQuery(String name)
- createNamedQuery(String name, Class<T> resultClass) TypedQuery
- createNativeQuery(String sqlString)
- createNativeQuery(String sqlString, Class resultClass)
- createNativeQuery(String sqlString, String resultSetMapping)
- createQuery(CriteriaQuery<T> criteriaQuery) TypedQuery
- createQuery(String qlString)
- createQuery(String qlString, Class<T> resultClass) TypedQuery
- detach(Object entity)
- equals(Object obj)
- find(Class<T> entityClass, Object primaryKey)
- find(Class<T> entityClass, Object primaryKey, LockModeType lockMode)

```
- Output Window:** Shows "WebBankingApp (run)" with a red error icon.

You have now:

```

44         out.println("</head>");
45         out.println("<body>");
46         out.println("<h1>Servlet BankingServlet at " + request.getContextPath());
47
48         Customer cust = emf.createEntityManager().createNamedQuery(hull);
49

```

Go to the EJB Customer.java and copy a query name as follows:

```

Source Packages
├── db
│   ├── Account.java
│   ├── Accountcustomer.java
│   ├── AccountcustomerPK.java
│   └── Customer.java
├── server
│   └── BankingServlet.java
├── Test Packages
├── Libraries
├── Test Libraries
└── Configuration Files

```

```

12  import javax.persistence.Id;
13  import javax.persistence.NamedQueries;
14  import javax.persistence.NamedQuery;
15  import javax.persistence.Table;
16
17  /**
18   *
19   * @author Geni
20   */
21  @Entity
22  @Table(name = "customer")
23  @NamedQueries({
24      @NamedQuery(name = "Customer.findAll", query = "SELECT c FROM Customer c"),
25      @NamedQuery(name = "Customer.findByIdCustomer", query = "SELECT c FROM Cust
26      @NamedQuery(name = "Customer.findByName", query = "SELECT c FROM Customer c
27      @NamedQuery(name = "Customer.findBySurname", query = "SELECT c FROM Custome
28  public class Customer implements Serializable {
29      private static final long serialVersionUID = 1L;
30      private

```

Copy "Customer.findAll"

```

@Entity
@Table(name = "customer")
@NamedQueries({
    @NamedQuery(name = "Customer.findAll", query = "SELECT c FROM Customer c"),
    @NamedQuery(name = "Customer.findByIdCustomer", query = "SELECT c FROM Custom
    @NamedQuery(name = "Customer.findByName", query = "SELECT c FROM Customer c W
    @NamedQuery(name = "Customer.findBySurname", query = "SELECT c FROM Customer
public class Customer implements Serializable {
    private static final long serialVersionUID = 1L;
    @Td

```

Paste what you copied from Customer, in the following window in the servlet. Now get the results from the database as follows:

```

response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
try {
    // TODO output your page here
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet BankingServlet</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Servlet BankingServlet at " + re

```

Returns:

a list of the results

Throws:

- IllegalStateException - if called for a Java Persistence query language UPDATE or DELETE statement
- QueryTimeoutException - if the query execution exceeds the query timeout value set and only the statement is rolled back
- TransactionRequiredException - if a lock mode has been set and there is no transaction
- RepositoryLockException - if pessimistic locking fails and the

```
Customer cust = emf.createEntityManager().createNamedQuery("Customer.findAll").
```

```

out.println("</body>");
out.println("</html>");

```

```

} finally {
    out.close();
}
}

```

HttpServlet methods. Click on the + sign on the left to edit

equals(Object obj)	boolean
executeUpdate()	int
getClass()	Class<?>
getFirstResult()	int
getFlushMode()	FlushModeType
getHints()	Map<String, Object>
getLockMode()	LockModeType
getMaxResults()	int
getParameter(String name)	Parameter<?>
getParameter(int position)	Parameter<?>
getParameter(String name, Class<T> type)	Parameter<T>
getParameter(int position, Class<T> t...)	Parameter<T>
getParameterValue(Parameter<T> param)	T
getParameterValue(String name)	Object
getParameterValue(int position)	Object
getParameters()	Set<Parameter<?>>
getResultList()	List

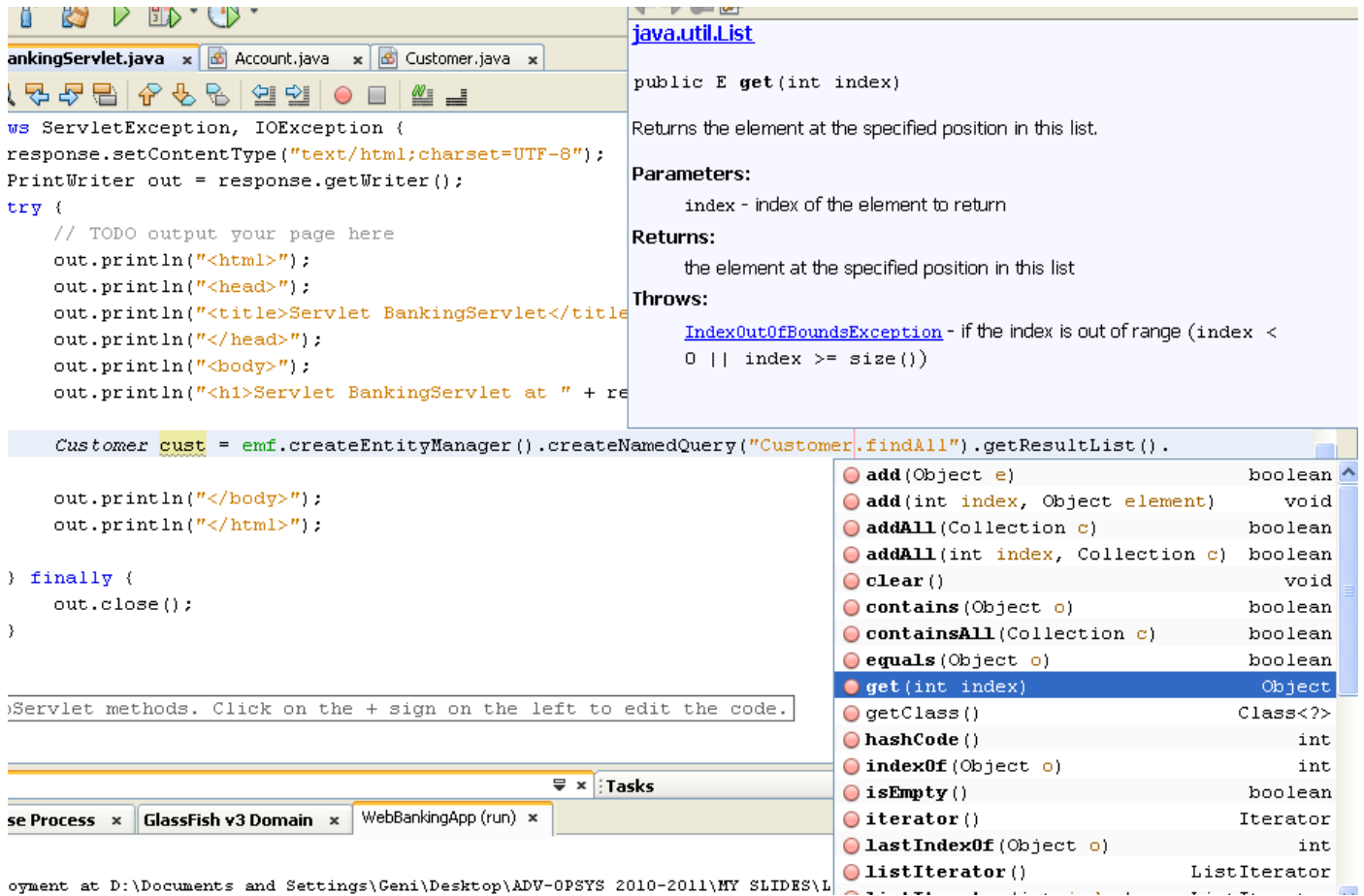
DB Database Process x GlassFish v3 Domain x WebBankingApp (run) x Tasks

```

...
.le-jsp:
.ace deployment at D:\Documents and Settings\Geni\Desktop\ADV-OPSYS 2010-2
.alizing...

```

Take the first record with get(0) as follows:



Now the code is:

```

out.println("<h1>Servlet BankingServlet at " + request.getContextPath () + "</h1>");

Customer cust = emf.createEntityManager ().createNamedQuery("Customer.findAll").getResultList ().get (0);

out.println("</body>");
out.println("</html>");

```

You still need to cast as follows by adding (Customer) before the statement::

```

out.println("<h1>Servlet BankingServlet at " + request.getContextPath () + "</h1>");

Customer cust = (Customer)emf.createEntityManager ().createNamedQuery("Customer.findAll").getResultList ().get (0);

out.println("</body>");
out.println("</html>");

```

Now arrange the imports as follows:

```

40 // TODO output your page here
41 out.println("<html>");
42 ("<head>");
43 ("<title>Servlet BankingServlet</title>");
44 ("</head>");
45 ("<body>");
46 ("<h1>Servlet BankingServlet at " + request.getContextPath () + "</h1>");
47
48 Customer cust = (Customer) emf.createEntityManager ().createNamedQuery ("Customer.findAll").getResultList ().get (0)
49
50 out.println("</html>");
51

```

cannot find symbol
symbol: class Customer
location: class server.BankingServlet

cannot find symbol
symbol: class Customer
location: class server.BankingServlet

Add import for db.Customer
Create class "Customer" in package server

Now we need to print the data on the web page that is going to be generated by the servlet. Write the following code as shown below to get the name of the customer:

```

7 throws ServletException, IOException {
8     response.setContentType ("text/html;charset=UTF-8");
9     PrintWriter out = response.getWriter ();
10    try {
11        // TODO output your page here
12        out.println("<html>");
13        out.println("<head>");
14        out.println("<title>Servlet BankingServlet</title>");
15        out.println("</head>");
16        out.println("<body>");
17        out.println("<h1>Servlet BankingServlet at " + request.getContextPath () + "</h1>");
18
19        Customer cust = (Customer) emf.createEntityManager ().createNamedQuery ("Customer.findAll").getResultList ().get (0)
20        out.println("<h2> The name of the customer is: " + cust.getName () + "</h2>");
21
22        out.println("</body>");
23        out.println("</html>");
24    } finally {
25        out.close ();
26    }
27 }

```

db.Customer

```

public String getName ()

```

Javadoc not found. Either Javadoc documentation for this item does not exist or you have not added specified Javadoc in the Java Platform Manager or the Library Manager.

- getName () String
- getSurname () String
- toString () String
- equals (Object object) boolean
- getClass () Class<?>
- getIdCustomer () Integer
- hashCode () int
- notify () void
- notifyAll () void
- setIdCustomer (Integer idCustomer) void
- setName (String name) void
- setSurname (String surname) void
- wait () void
- wait (long timeout) void
- wait (long timeout, int nanos) void

Java DB Database Process x GlassFish v3 Domain x WebBankingApp (run) x

Browsing: http://localhost:8080/WebBankingApp/

run-display-browser:

run:

Now the code looks as follows:

```

Customer cust = (Customer) emf.createEntityManager ().createNamedQuery ("Customer.findAll").getResultList ().get (0);
out.println("<h2> The name of the customer is: " + cust.getName () + "</h2>");

```

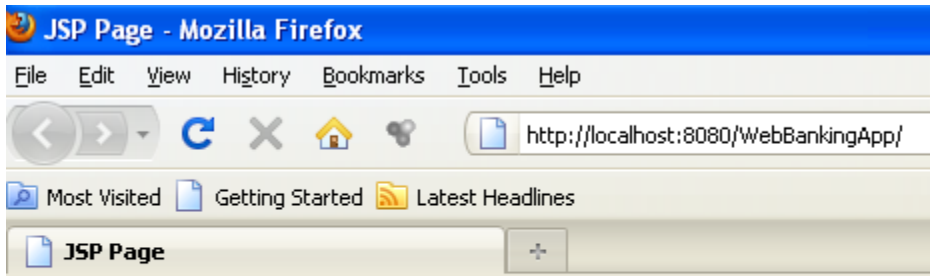
If you want to take also the surname add as follows:

```

Customer cust = (Customer) emf.createEntityManager ().createNamedQuery ("Customer.findAll").getResultList ().get (0);
out.println("<h2> The name of the customer is: " + cust.getName () + "</h2>");
out.println("<h2> The surname of the customer is: " + cust.getSurname () + "</h2>");

```

Save the project and run it.

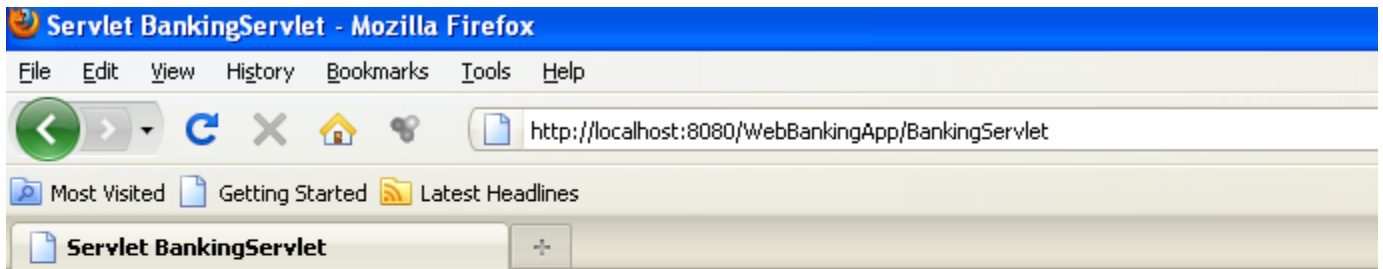


Hello World!

Add the servlet name in the browser:

<http://localhost:8080/WebBankingApp/BankingServlet>

The following will appear:



Servlet BankingServlet at /WebBankingApp

The name of the customer is: Leo

The surname of the customer is: Messi

If we want to take some data about the accounts:

Go to the EJB Account.java and copy a query name as follows:

```

12 import javax.persistence.Id;
13 import javax.persistence.NamedQueries;
14 import javax.persistence.NamedQuery;
15 import javax.persistence.Table;
16
17 /**
18  *
19  * @author Geni
20  */
21 @Entity
22 @Table(name = "account")
23 @NamedQueries({
24     @NamedQuery(name = "Account.findAll", query = "SELECT a FROM Account a"),
25     @NamedQuery(name = "Account.findByIdAccount", query = "SELECT a FROM Account a WHERE a.id = ?1"),
26     @NamedQuery(name = "Account.findByBalance", query = "SELECT a FROM Account a WHERE a.balance = ?1")
27 })
28 public class Account implements Serializable {
29     private static final long serialVersionUID = 1L;
30     @Id

```

Copy "Account"

```

@Entity
@Table(name = "account")
@NamedQueries({
    @NamedQuery(name = "Account.findAll", query = "SELECT a FROM Account a"),
    @NamedQuery(name = "Account.findByIdAccount", query = "SELECT a FROM Account a WHERE a.id = ?1"),
    @NamedQuery(name = "Account.findByBalance", query = "SELECT a FROM Account a WHERE a.balance = ?1")
})
public class Account implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id

```

Add the following code:

```

Customer cust = (Customer) emf.createEntityManager().createNamedQuery("Customer.findAll").getResultList().get(0);
out.println("<h2> The name of the customer is: " + cust.getName() + "</h2>");
out.println("<h2> The surname of the customer is: " + cust.getSurname() + "</h2>");
// Variable account is not used
Account account = (Account) emf.createEntityManager().createNamedQuery("Account.findAll").getResultList().get(0);

```

Arrange the imports by click the red point on the left shown by NetBeans:

```

44         out.println("<title>Servlet Bankin
45         out.println("</head>");
46         out.println("<body>");
47         ("<h1>Servlet BankingSe
48         symbol: class Account
49         location: class server.BankingServlet st = (Customer) emf.crea
50         ("<h2> The name of the
51         symbol: class Account ("<h2> The surname of t
52         location: class server.BankingServlet
53         Account account = (Account) emf.crea
54         Add import for db.Account
55         Create class "Account" in package server
56         out.println("</body>");
57         out.println("</html>");
58
59     } finally {

```

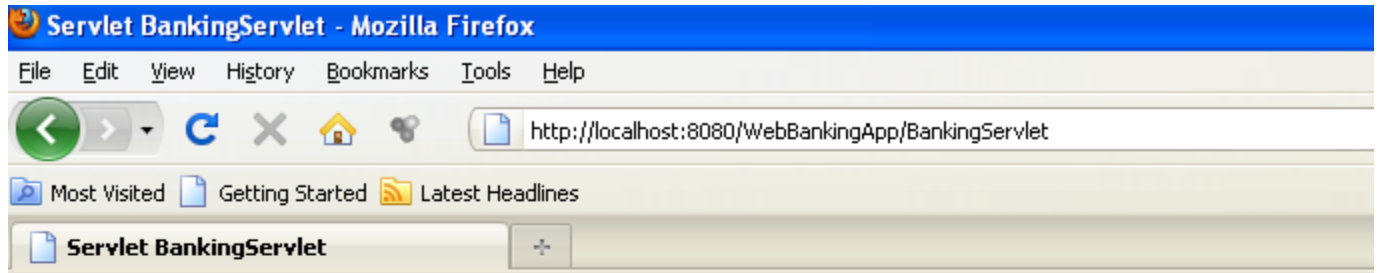
Now add the code to the id of the account and the balance as follows:

```

Account account = (Account) emf.createEntityManager().createNamedQuery("Account.findAll").getResultList().get(0);
out.println("<h2> The ID of the account is: " + account.getIdAccount() + "</h2>");
out.println("<h2> The balance of the account is: " + account.getBalance() + "</h2>");

```

Save the project. If you refresh the browser you will get:



Servlet BankingServlet at /WebBankingApp

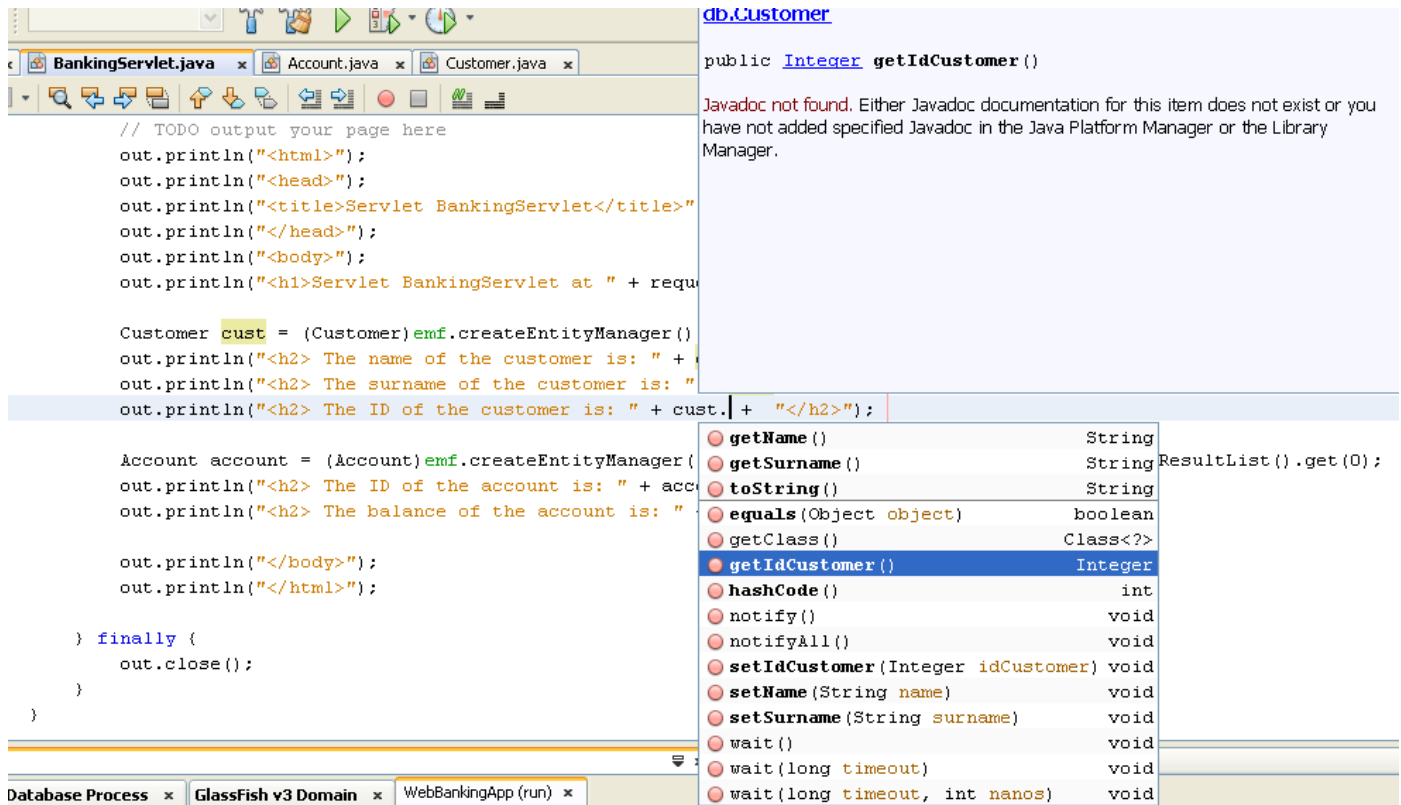
The name of the customer is: Leo

The surname of the customer is: Messi

The ID of the account is: 1

The balance of the account is: 580436.0

We can also add the ID of the customer as follows:



The code is:

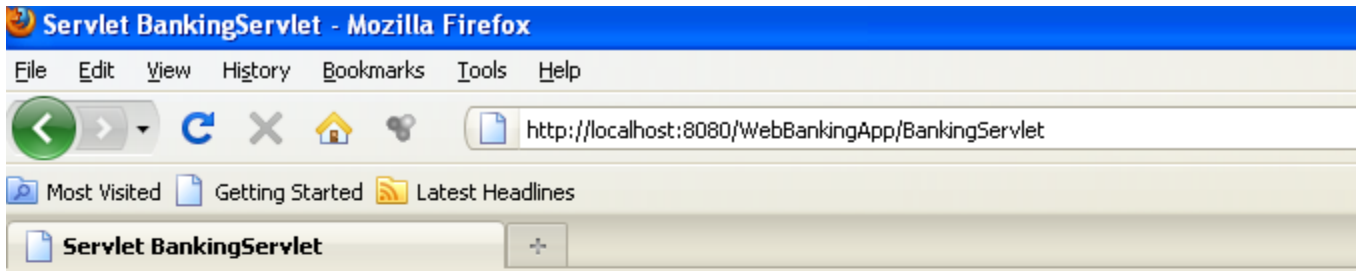
```

Customer cust = (Customer)emf.createEntityManager().createNamedQuery("Customer.findAll").getResultList().get(0);
out.println("<h2> The name of the customer is: " + cust.getName() + "</h2>");
out.println("<h2> The surname of the customer is: " + cust.getSurname() + "</h2>");
out.println("<h2> The ID of the customer is: " + cust.getIdCustomer() + "</h2>");

Account account = (Account)emf.createEntityManager().createNamedQuery("Account.findAll").getResultList().get(0);
out.println("<h2> The ID of the account is: " + account.getIdAccount() + "</h2>");
out.println("<h2> The balance of the account is: " + account.getBalance() + "</h2>");

```

If you refresh the browser you will get:



Servlet BankingServlet at /WebBankingApp

The name of the customer is: Leo

The surname of the customer is: Messi

The ID of the customer is: 1

The ID of the account is: 1

The balance of the account is: 580436.0

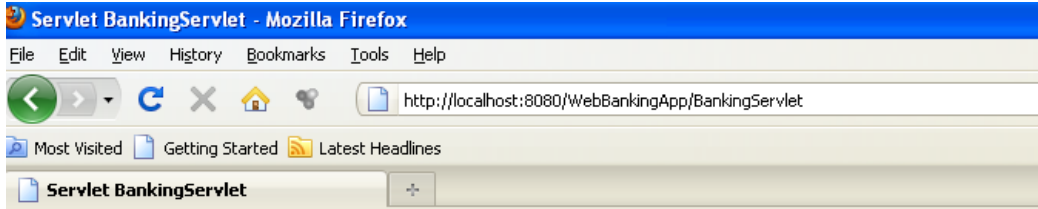
If we want to add the relationship between the customer and the account add the following:

```
Customer cust = (Customer)emf.createEntityManager().createNamedQuery("Customer.findAll").getResultList().get(0);
out.println("<h2> The name of the customer is: " + cust.getName() + "</h2>");
out.println("<h2> The surname of the customer is: " + cust.getSurname() + "</h2>");
out.println("<h2> The ID of the customer is: " + cust.getIdCustomer() + "</h2>");

Accountcustomer accCust = (Accountcustomer)emf.createEntityManager().createNamedQuery("Accountcustomer.findAll").getResultList().get(0);
out.println("<h2> The customer with account ID: " + accCust.getAccountcustomerPK().getIdAccount() + " has the ID: " +
    accCust.getAccountcustomerPK().getIdCustomer() + "</h2>");

Account account = (Account)emf.createEntityManager().createNamedQuery("Account.findAll").getResultList().get(0);
out.println("<h2> The ID of the account is: " + account.getIdAccount() + "</h2>");
out.println("<h2> The balance of the account is: " + account.getBalance() + "</h2>");
```

Refresh the browser and you will get the following:



Servlet BankingServlet at /WebBankingApp

The name of the customer is: Leo

The surname of the customer is: Messi

The ID of the customer is: 1

The customer with account ID: 1 has the ID: 1

The ID of the account is: 1

The balance of the account is: 580436.0

If you need to retrieve an account directly by ID you can use the named queries by setting the appropriate parameters:

Add the following code:

```
// how to get account by ID
Account account2 = (Account)emf.createEntityManager().createNamedQuery("Account.findByIdAccount").
    setParameter("idAccount", 2).getResultList().get(0);
out.println("<h2> Retrieving account by ID </h2>");
out.println("<h2> The ID of the account is: " + account2.getIdAccount() + "</h2>");
out.println("<h2> The balance of the account is: " + account2.getBalance() + "</h2>");
```

Check the setParameter method above which sets the parameter idAccount to the value 2.

The result of the query will be:

Servlet BankingServlet at /WebBankingApp

The name of the customer is: Leo

The surname of the customer is: Messi

The ID of the customer is: 1

The customer with account ID: 1 has the ID: 1

The ID of the account is: 1

The balance of the account is: 580436.0

Retreiving account by ID

The ID of the account is: 2

The balance of the account is: 3000.0

For more details on Creating Queries Using the Java Persistence Query Language you can refer the following Oracle resource:

<http://docs.oracle.com/javase/6/tutorial/doc/bnbrg.html>