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

Master of Science in Computer Science

Distributed Systems
Manual for Laboratory Practice

Enterprise JavaBeans

PART III
A Web Banking Application with EJB and MySQL
Development of Client Interface and Queries on Database

Prof. Dr. Marenglen Biba
Department of Computer Science
E-mail: marenglenbiba@unyt.edu.al

1. Document Purpose

This document contains explanations on how to run the following programs: RMI Servers, RMI Client and Database Server.

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=

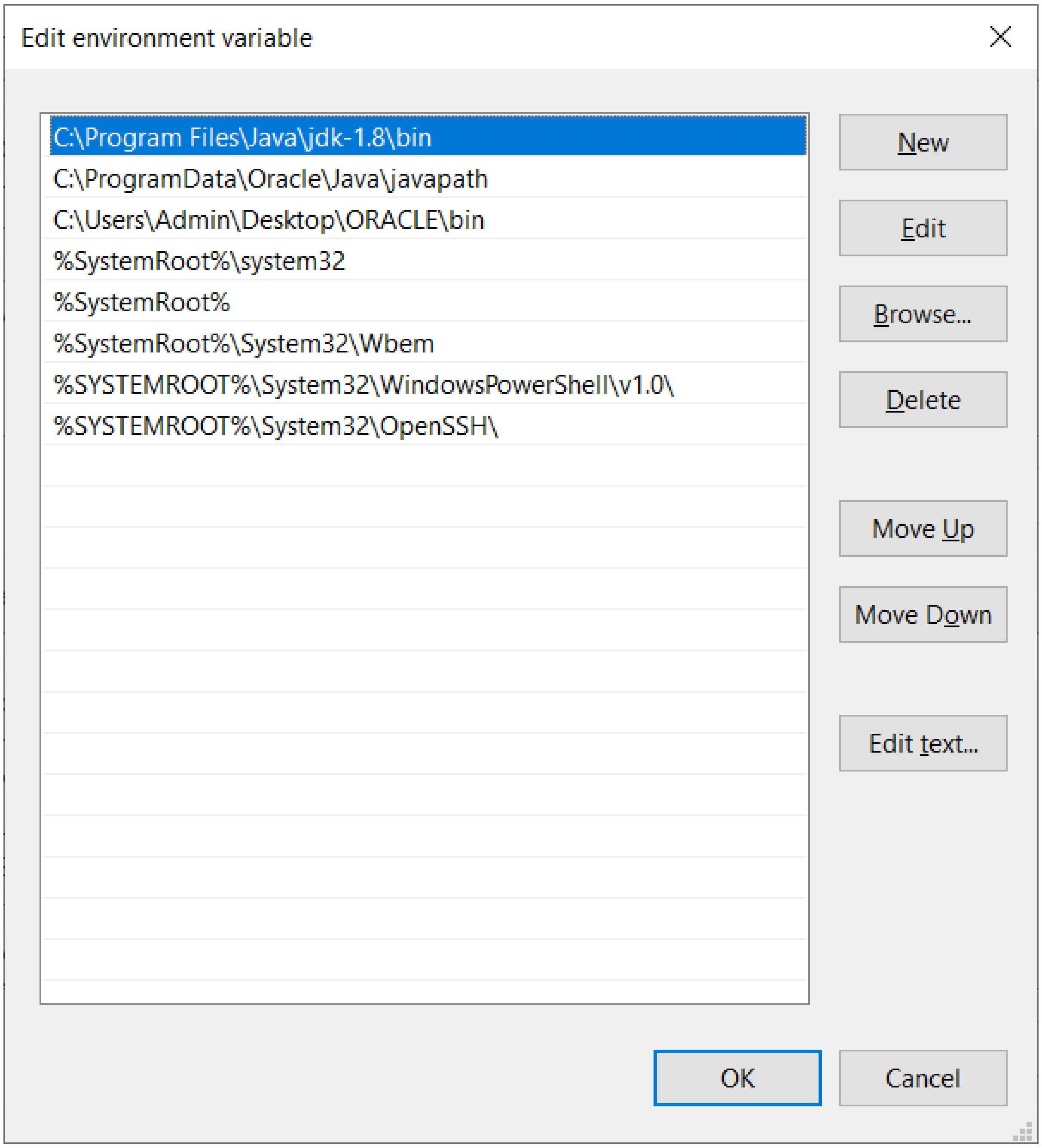
- Install MySQL 5.0 and MySQL WorkBench 8.0

Set path and Path variables in the operating system

Click on Enviroment Variables.

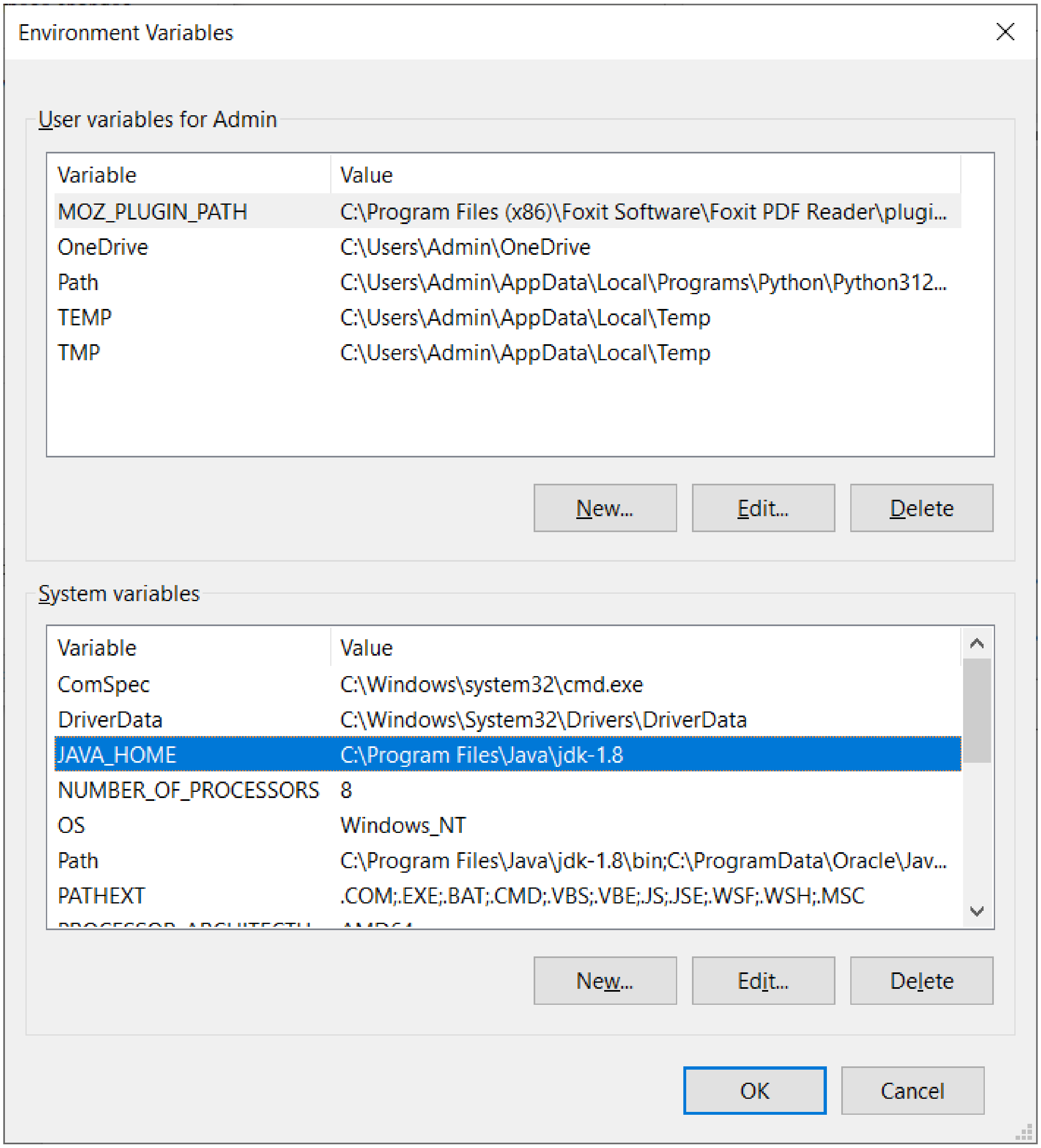
Find the Path system variable and click Edit. Set the value of the variable to the directory where you have installed Java, for example:

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



```
C:\Program Files\Java\jdk-1.8\bin
C:\ProgramData\Oracle\Java\javapath
C:\Users\Admin\Desktop\ORACLE\bin
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
```

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

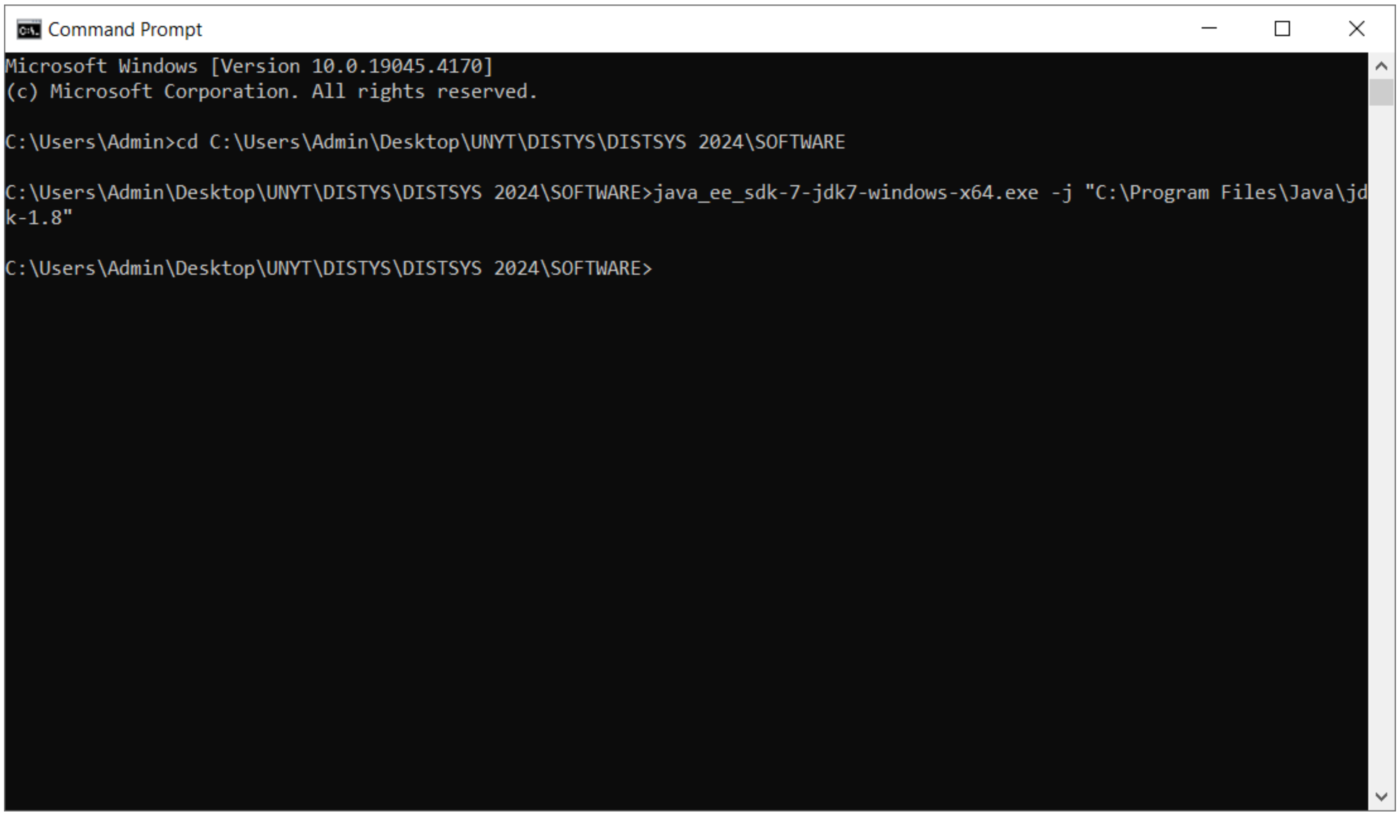
System variables

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

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

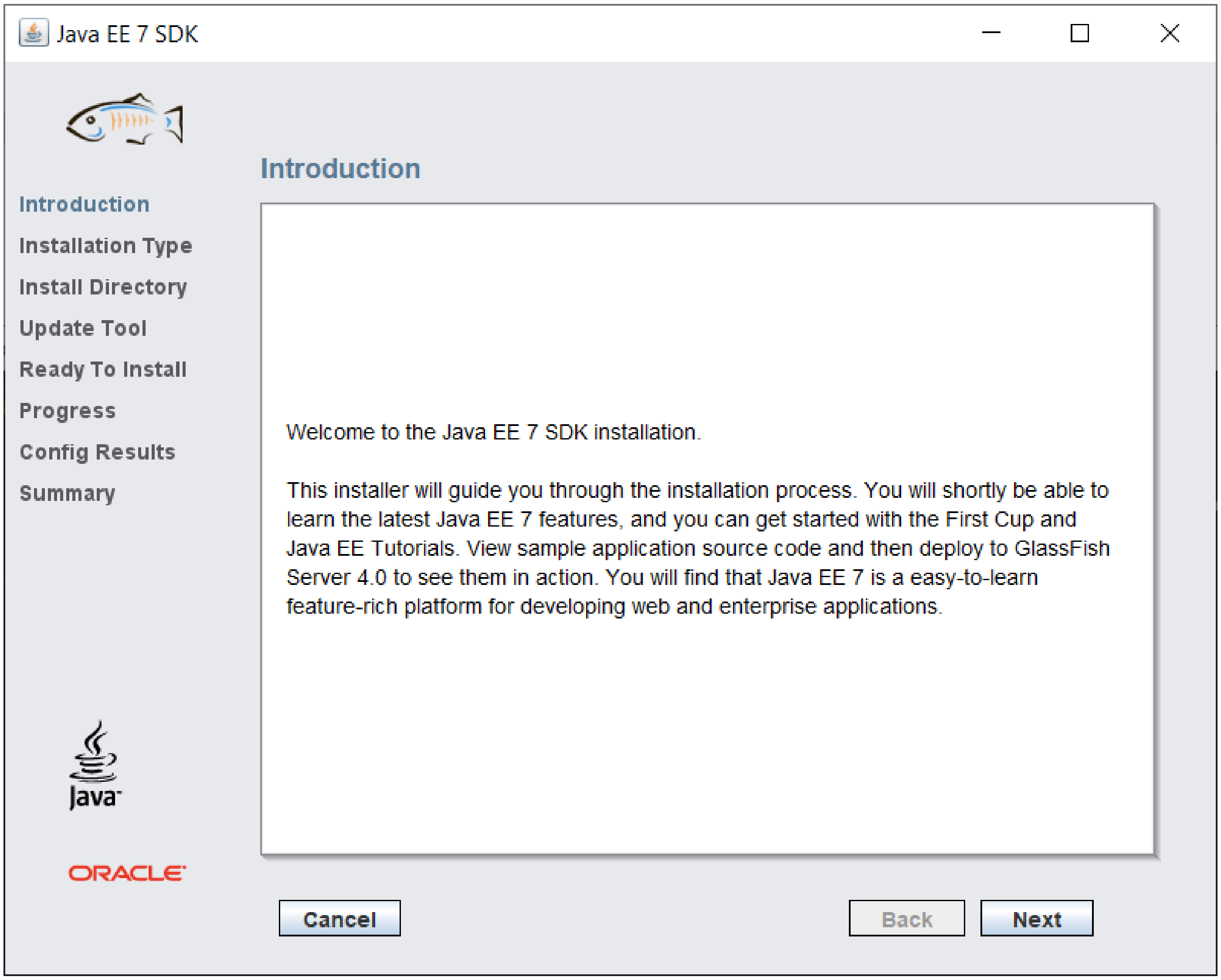
Download and install Java EE SDK.

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



Command Prompt

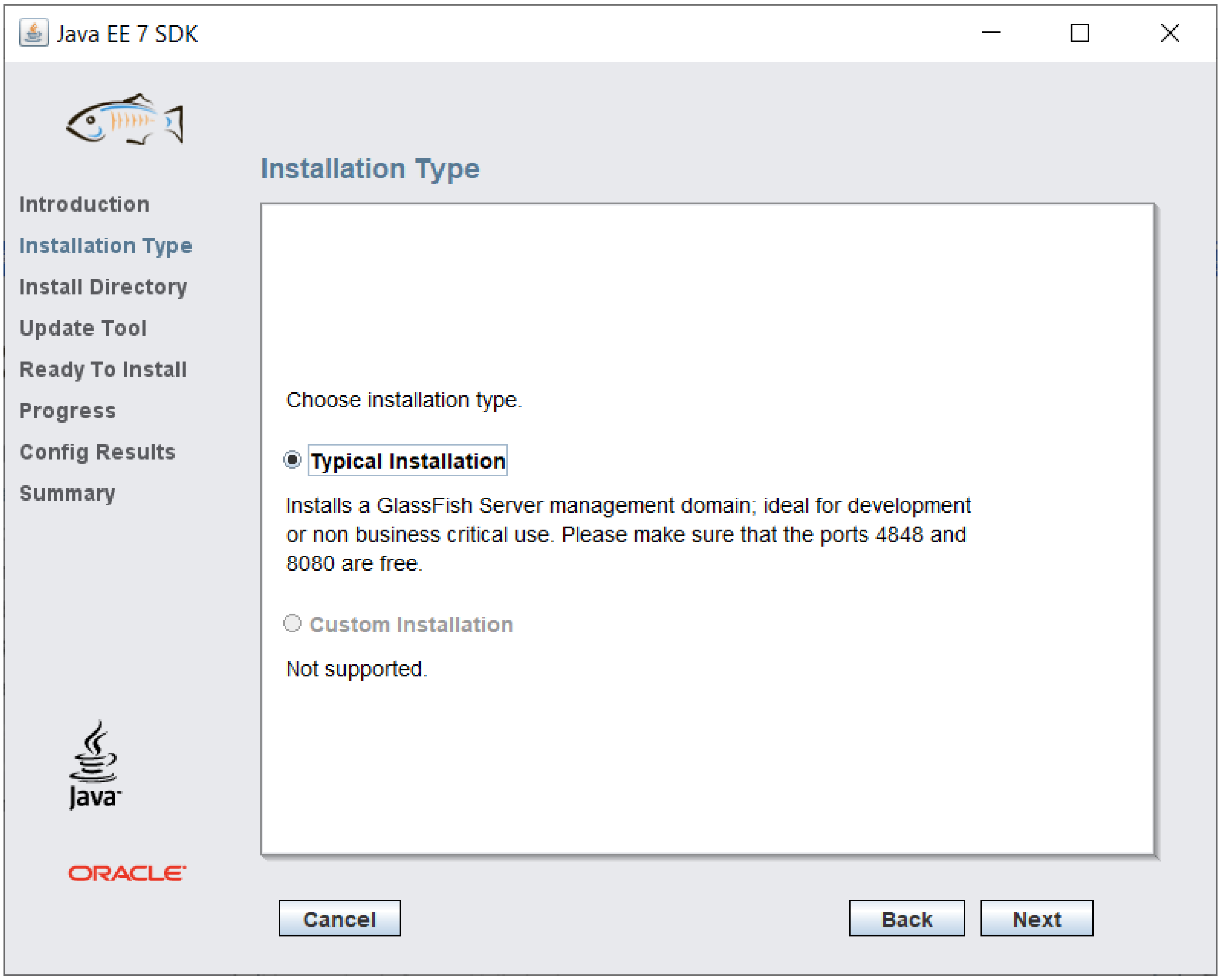
```
Microsoft Windows [Version 10.0.19045.4170]
(c) Microsoft Corporation. All rights reserved.

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\jd
k-1.8"

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



Introduction

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.



Installation Type

Choose installation type.

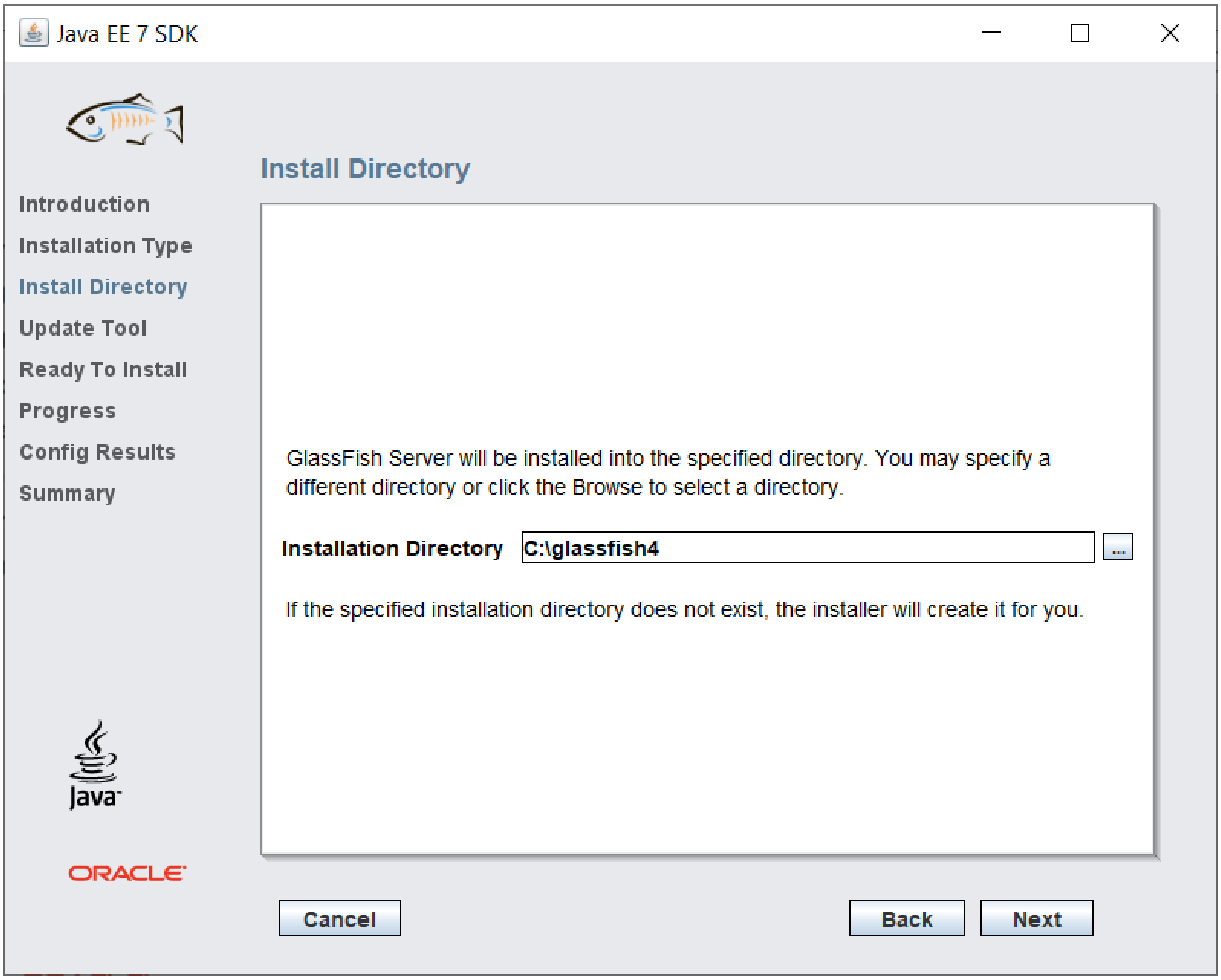
[✓ Typical Installation]

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.

Choose directory for Glassfish:



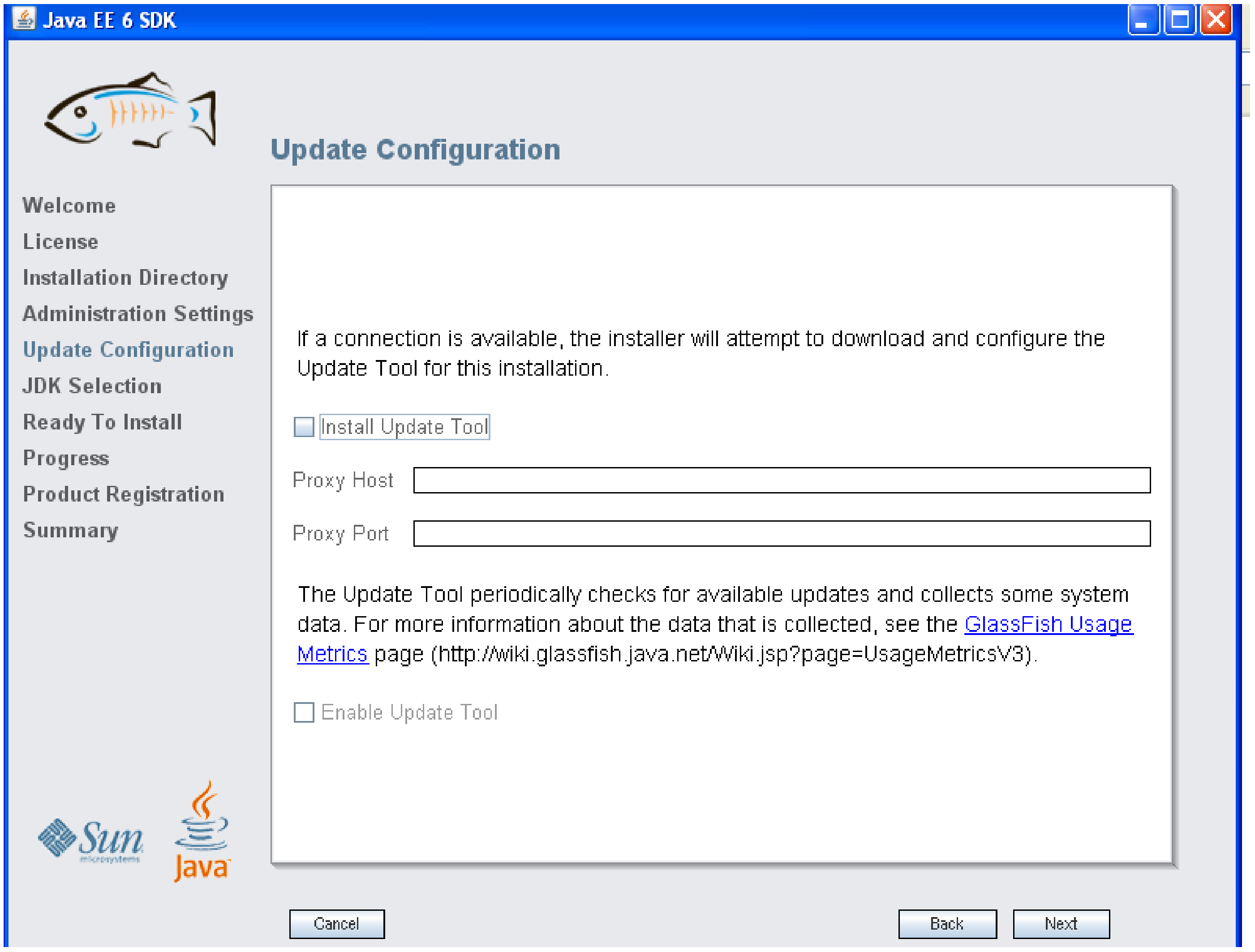
Install Directory

GlassFish Server will be installed into the specified directory. You may specify a different directory or click the Browse to select a directory.

Installation Directory C:\glassfish4

If the specified installation directory does not exist, the installer will create it for you.

Click again Next:



Update Configuration

If a connection is available, the installer will attempt to download and configure the Update Tool for this installation.

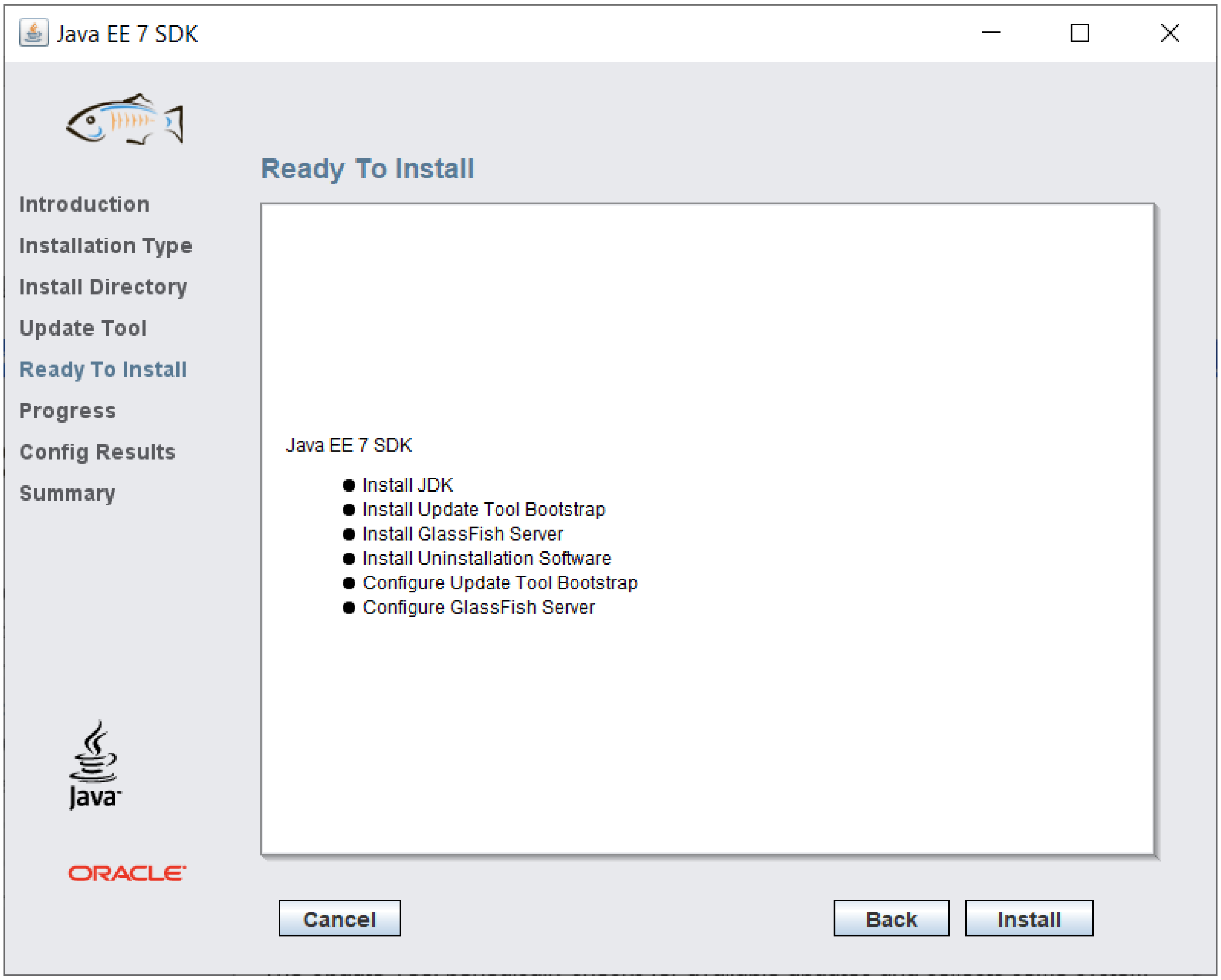
☐ Install Update Tool

Proxy Host

Proxy Port

The Update Tool periodically checks for available updates and collects some system data. For more information about the data that is collected, see the GlassFish Usage Metrics page (http://wiki.glassfish.java.net/Wiki.jsp?page=UsageMetricsV3).

☐ Enable Update Tool

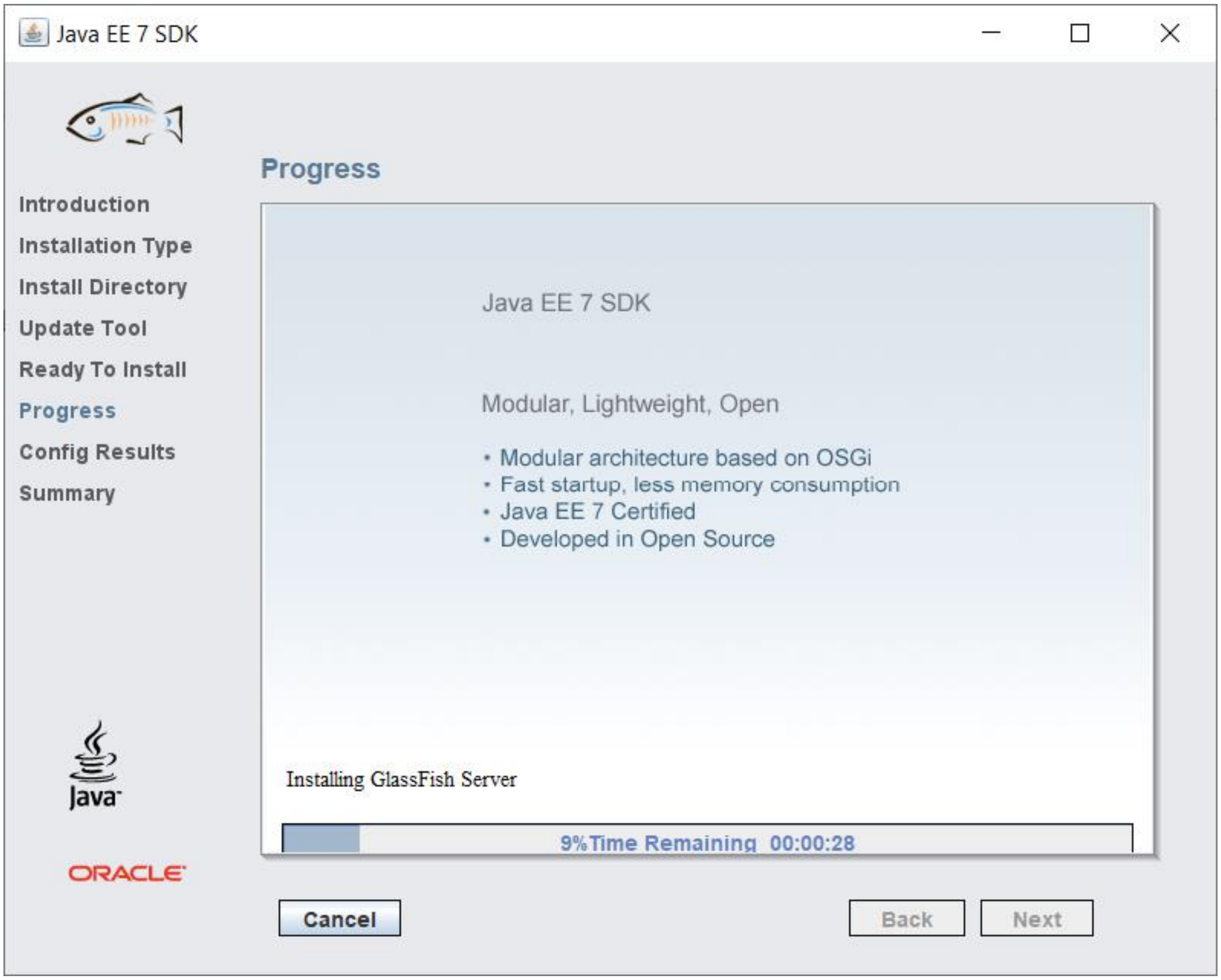


Java EE 7 SDK

Ready To Install

Java EE 7 SDK

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



Progress

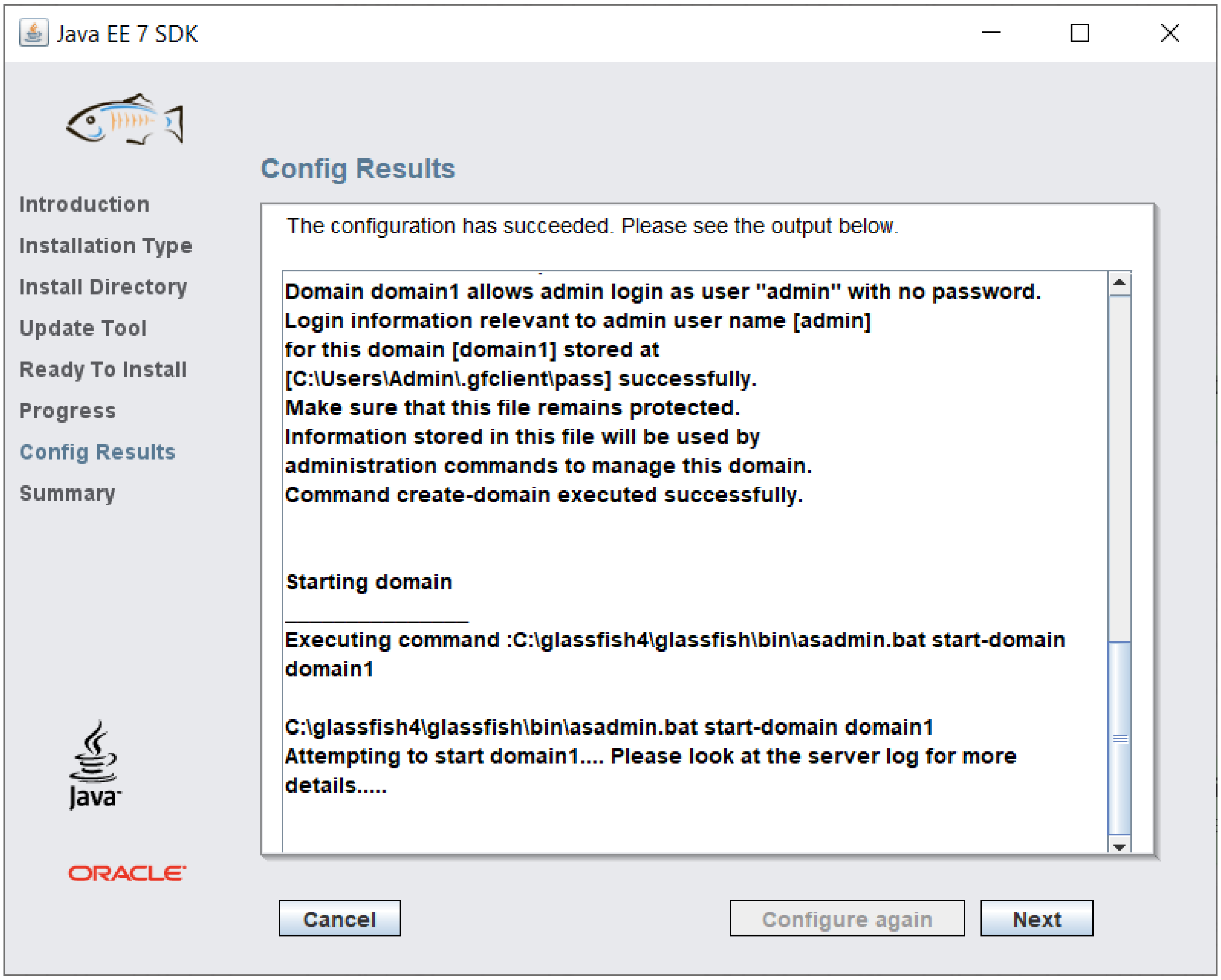
Java EE 7 SDK

Modular, Lightweight, Open

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

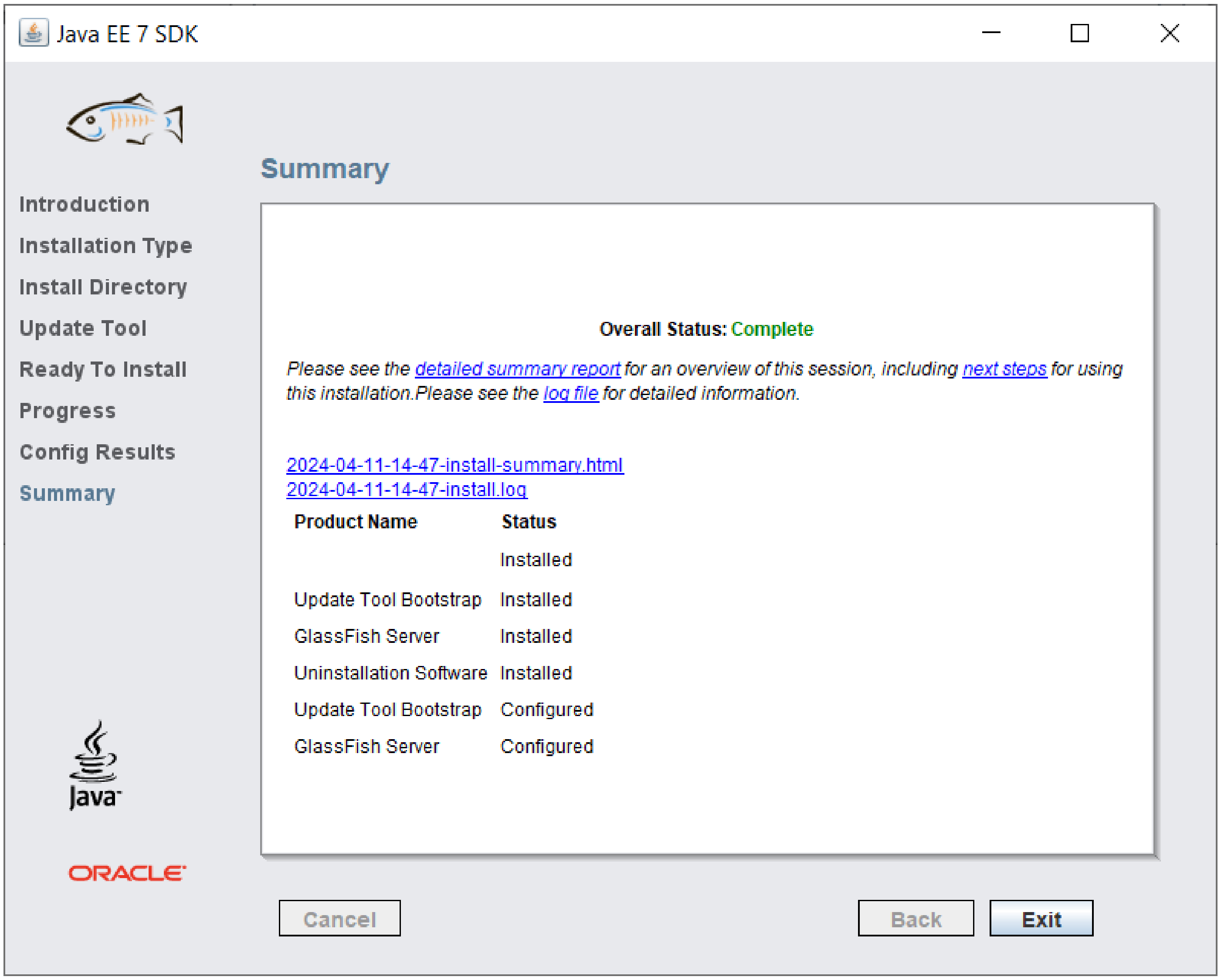
Installing GlassFish Server

9%Time Remaining 00:00:28



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.....
```



Summary

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

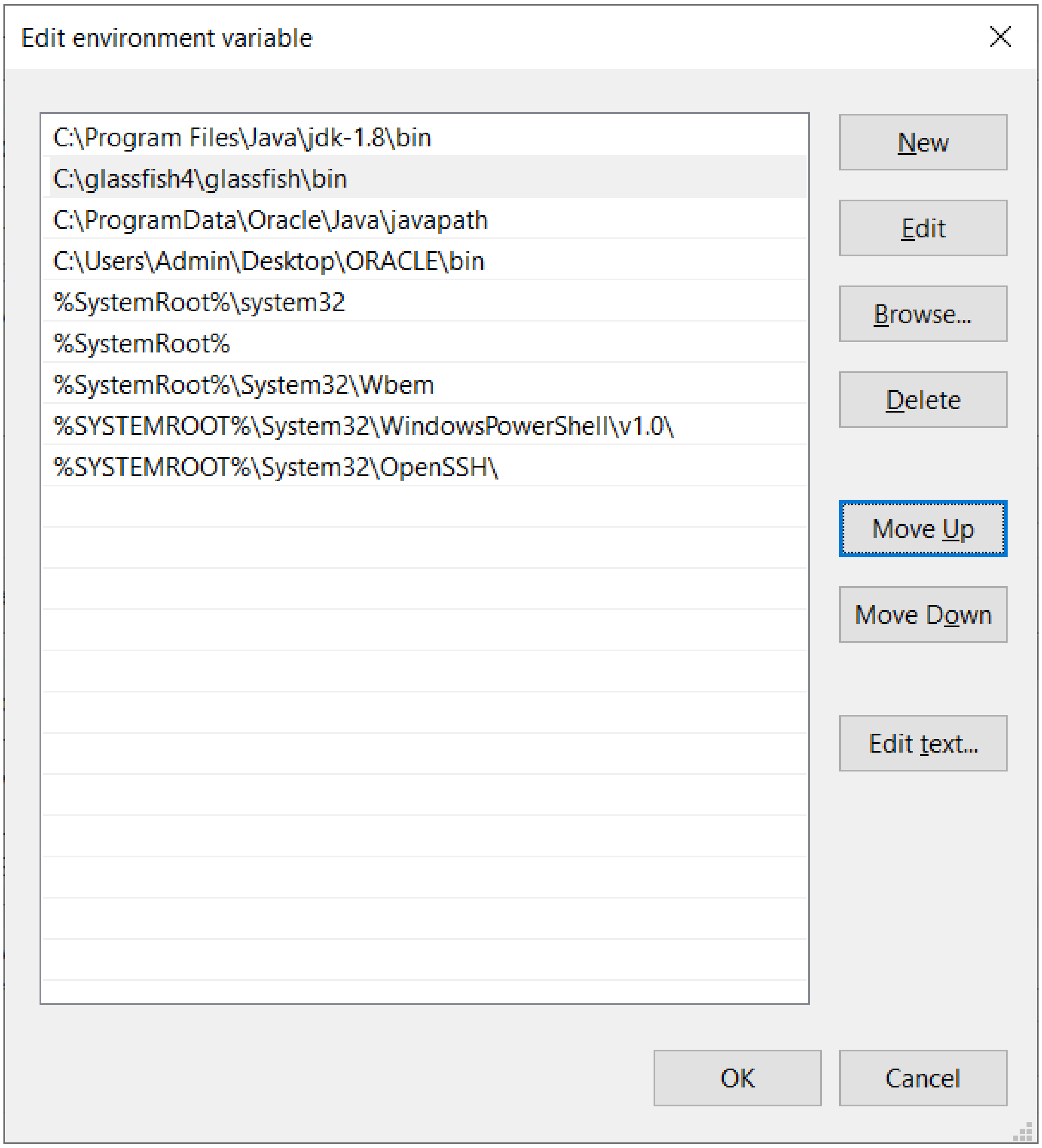
Product Name	Status
	Installed
Update Tool Bootstrap	Installed
GlassFish Server	Installed
Uninstallation Software	Installed
Update Tool Bootstrap	Configured
GlassFish Server	Configured

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.

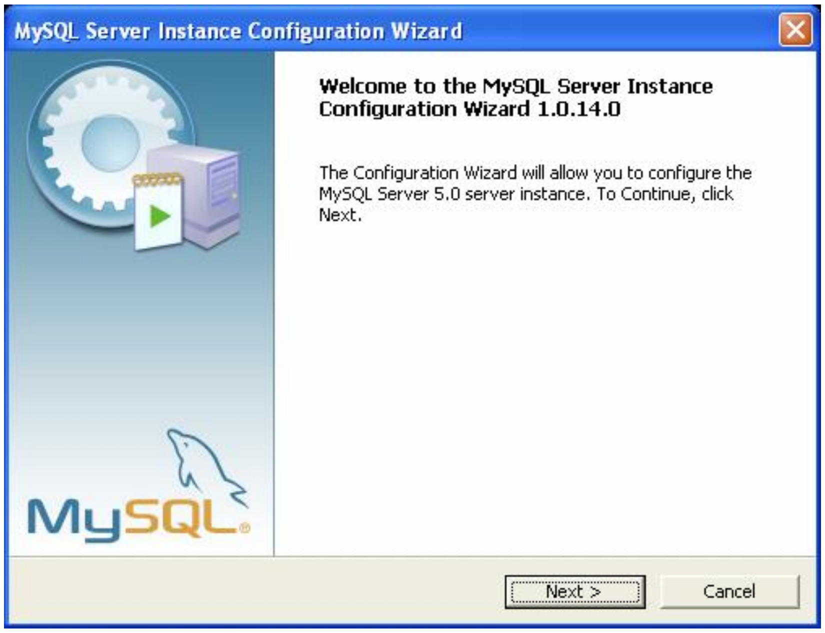

Edit environment variable

C:\Program Files\Java\jdk-1.8\bin
C:\glassfish4\glassfish\bin
C:\ProgramData\Oracle\Java\javapath
C:\Users\Admin\Desktop\ORACLE\bin
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\

In the variable value add the path of Glassfish.

2. Download and Install MySQL Server:

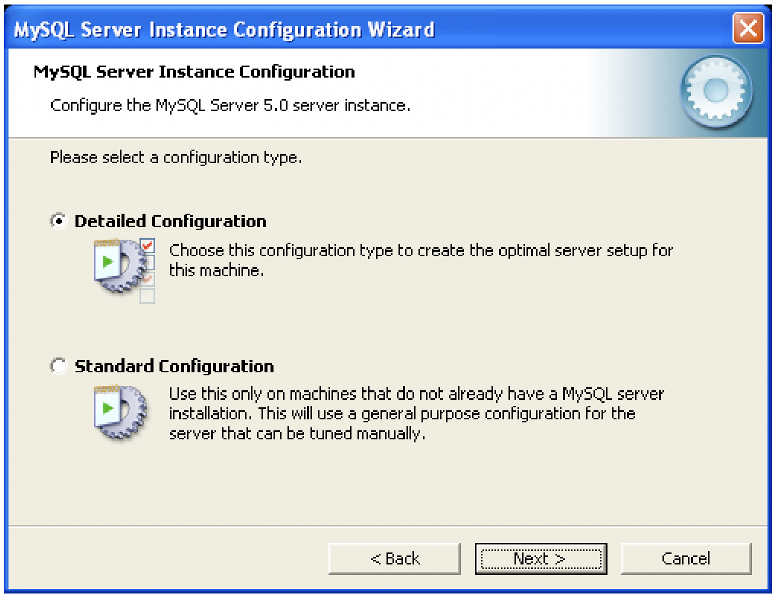
After you download use MySQL Server Instance Config Wizard



Welcome to the MySQL Server Instance Configuration Wizard 1.0.14.0

The Configuration Wizard will allow you to configure the MySQL Server 5.0 server instance. To Continue, click Next.

Choose the detailed configuration:



MySQL Server Instance Configuration

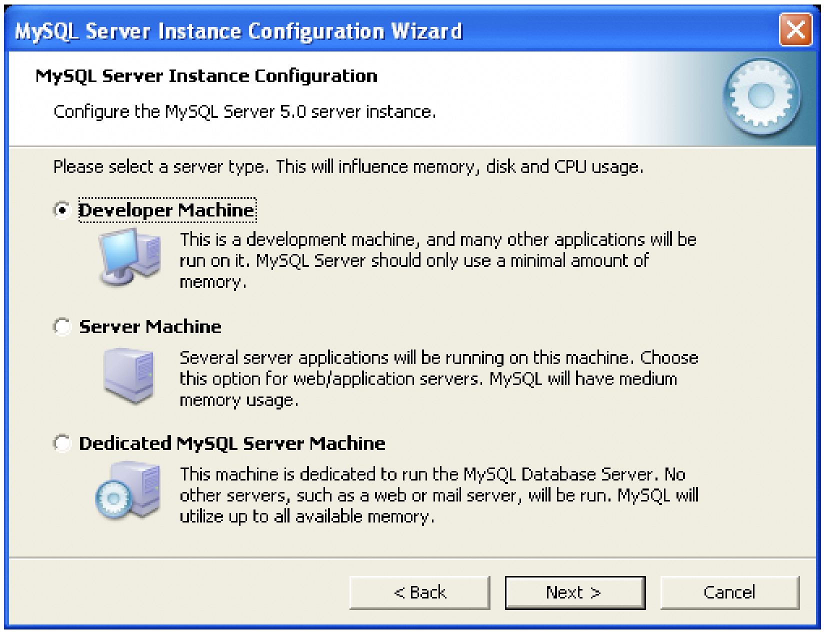
Configure the MySQL Server 5.0 server instance.

Please select a configuration type.

[✓ **Detailed Configuration**]
Choose this configuration type to create the optimal server setup for this machine.

Standard Configuration
Use this only on machines that do not already have a MySQL server installation. This will use a general purpose configuration for the server that can be tuned manually.

Choose developer machine:


MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 5.0 server instance.

Please select a server type. This will influence memory, disk and CPU usage.

[✓ Developer Machine]

This is a development machine, and many other applications will be run on it. MySQL Server should only use a minimal amount of memory.

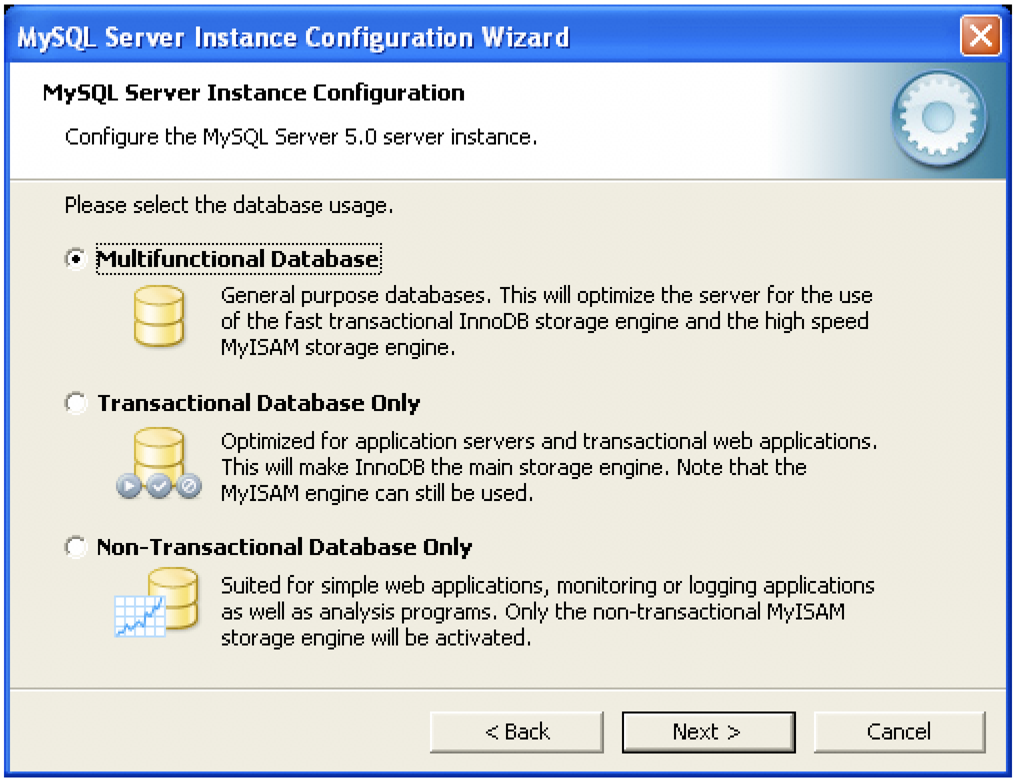
Server Machine

Several server applications will be running on this machine. Choose this option for web/application servers. MySQL will have medium memory usage.

Dedicated MySQL Server Machine

This machine is dedicated to run the MySQL Database Server. No other servers, such as a web or mail server, will be run. MySQL will utilize up to all available memory.

Choose multifunctional:


MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 5.0 server instance.

Please select the database usage.

[✓ Multifunctional Database]

General purpose databases. This will optimize the server for the use of the fast transactional InnoDB storage engine and the high speed MyISAM storage engine.

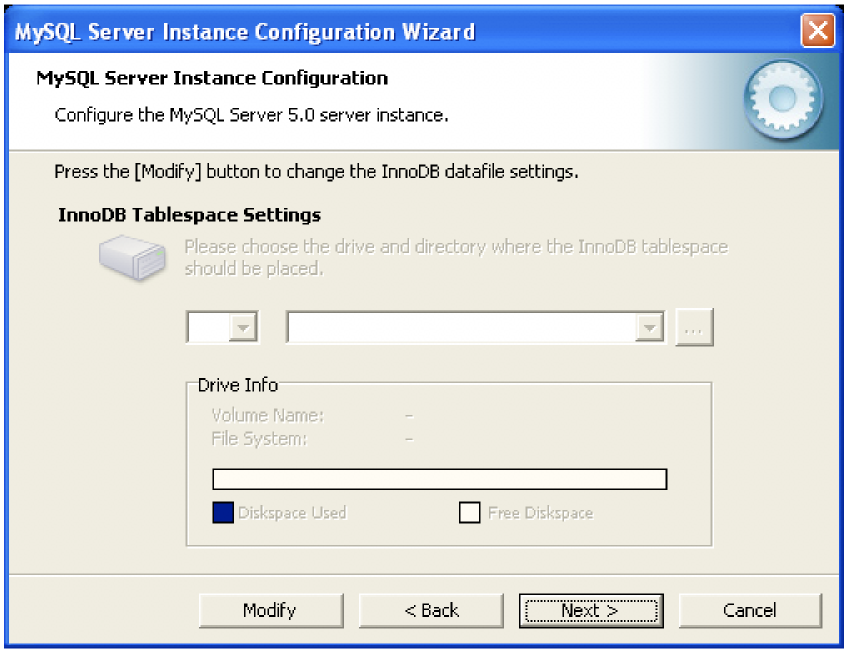
Transactional Database Only

Optimized for application servers and transactional web applications. This will make InnoDB the main storage engine. Note that the MyISAM engine can still be used.

Non-Transactional Database Only

Suited for simple web applications, monitoring or logging applications as well as analysis programs. Only the non-transactional MyISAM storage engine will be activated.

Choose Drive:



MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 5.0 server instance.

Press the [Modify] button to change the InnoDB datafile settings.

InnoDB Tablespace Settings

Please choose the drive and directory where the InnoDB tablespace should be placed.

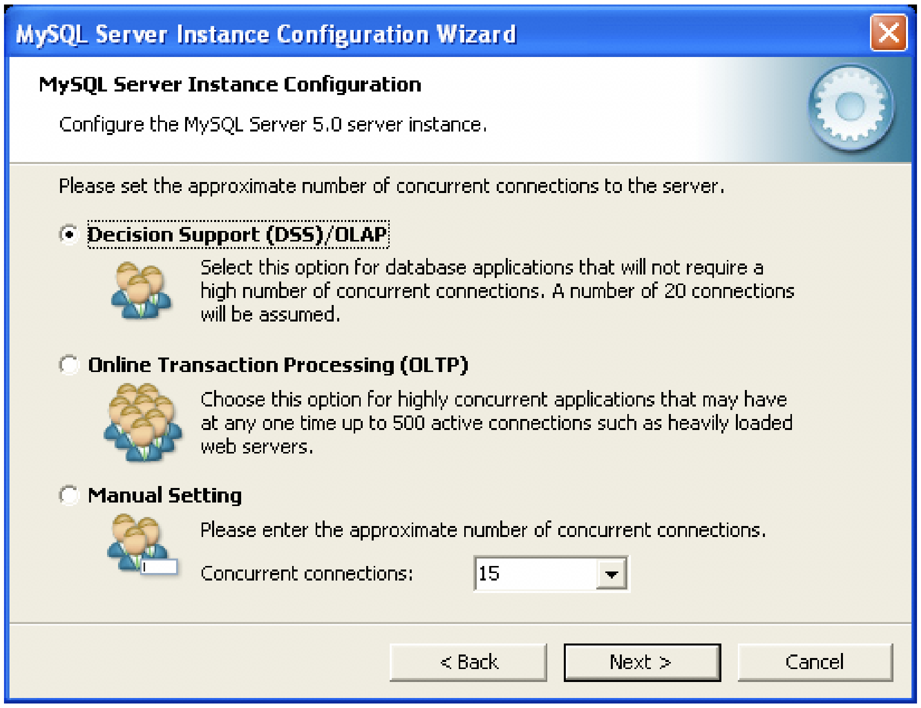
Drive Info

Volume Name: -

File System: -

■ Diskspace Used ☐ Free Diskspace

Choose decision support:



MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 5.0 server instance.

Please set the approximate number of concurrent connections to the server.

[✓ **Decision Support (DSS)/OLAP**]

Select this option for database applications that will not require a high number of concurrent connections. A number of 20 connections will be assumed.

Online Transaction Processing (OLTP)

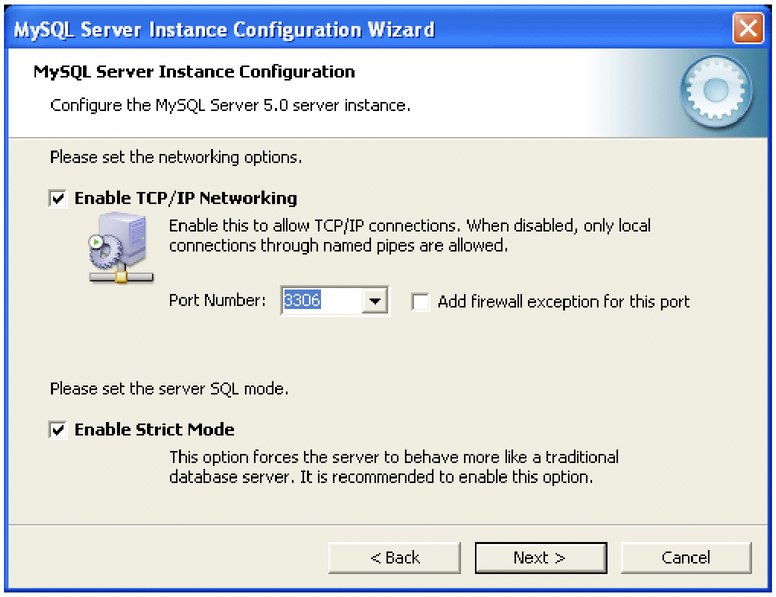
Choose this option for highly concurrent applications that may have at any one time up to 500 active connections such as heavily loaded web servers.

Manual Setting

Please enter the approximate number of concurrent connections.

Concurrent connections: 15

Perform the following checks:


MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 5.0 server instance.

Please set the networking options.

[✓] **Enable TCP/IP Networking**

Enable this to allow TCP/IP connections. When disabled, only local connections through named pipes are allowed.

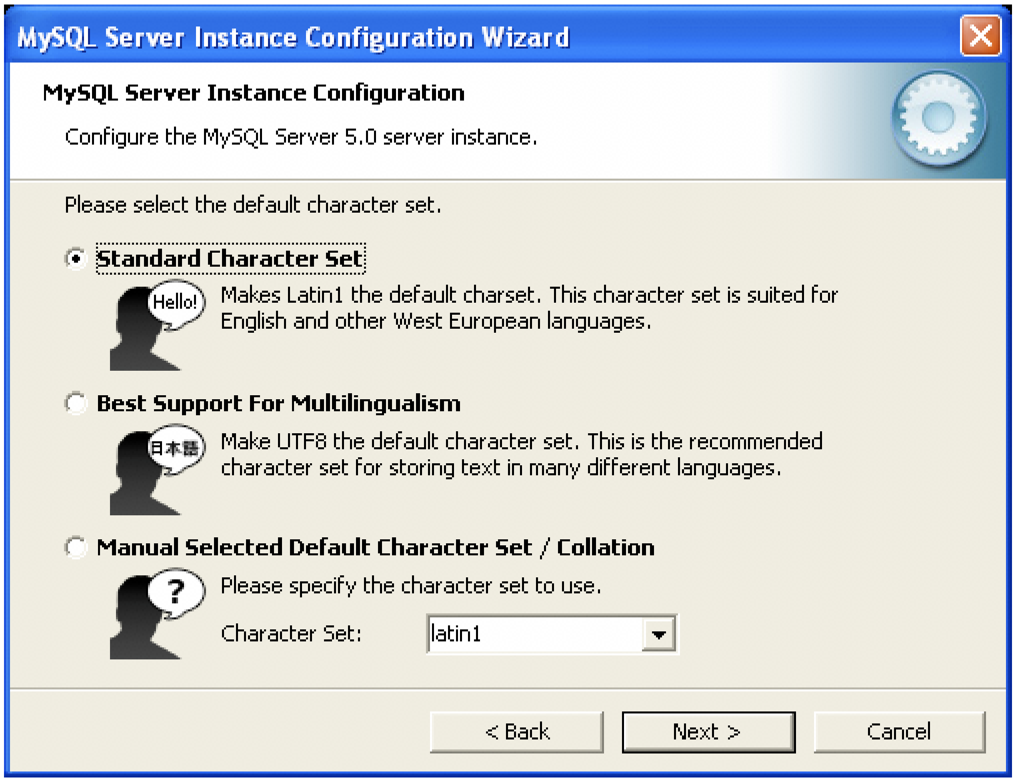
Port Number: 3306 [ ] Add firewall exception for this port

Please set the server SQL mode.

[✓] **Enable Strict Mode**

This option forces the server to behave more like a traditional database server. It is recommended to enable this option.

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.


MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 5.0 server instance.

Please select the default character set.

[✓ **Standard Character Set**]

Makes Latin1 the default charset. This character set is suited for English and other West European languages.

Best Support For Multilingualism

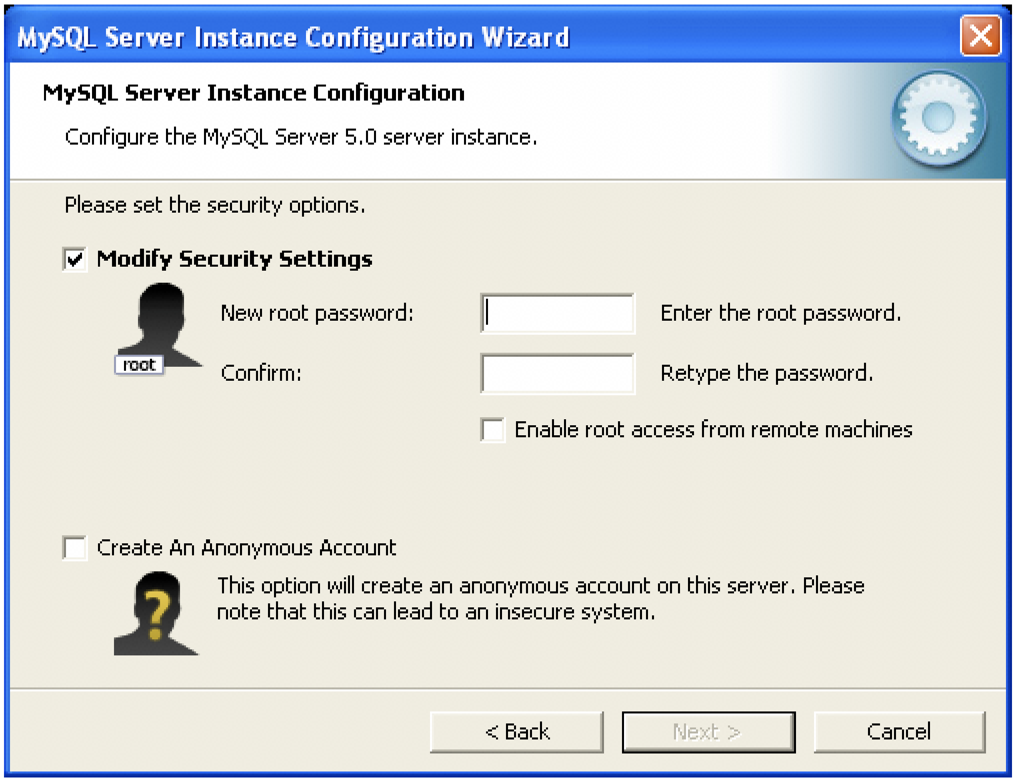
Make UTF8 the default character set. This is the recommended character set for storing text in many different languages.

Manual Selected Default Character Set / Collation

Please specify the character set to use.

Character Set: latin1

Set the password for root:


MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 5.0 server instance.

Please set the security options.

[✓] **Modify Security Settings**

New root password: Enter the root password.

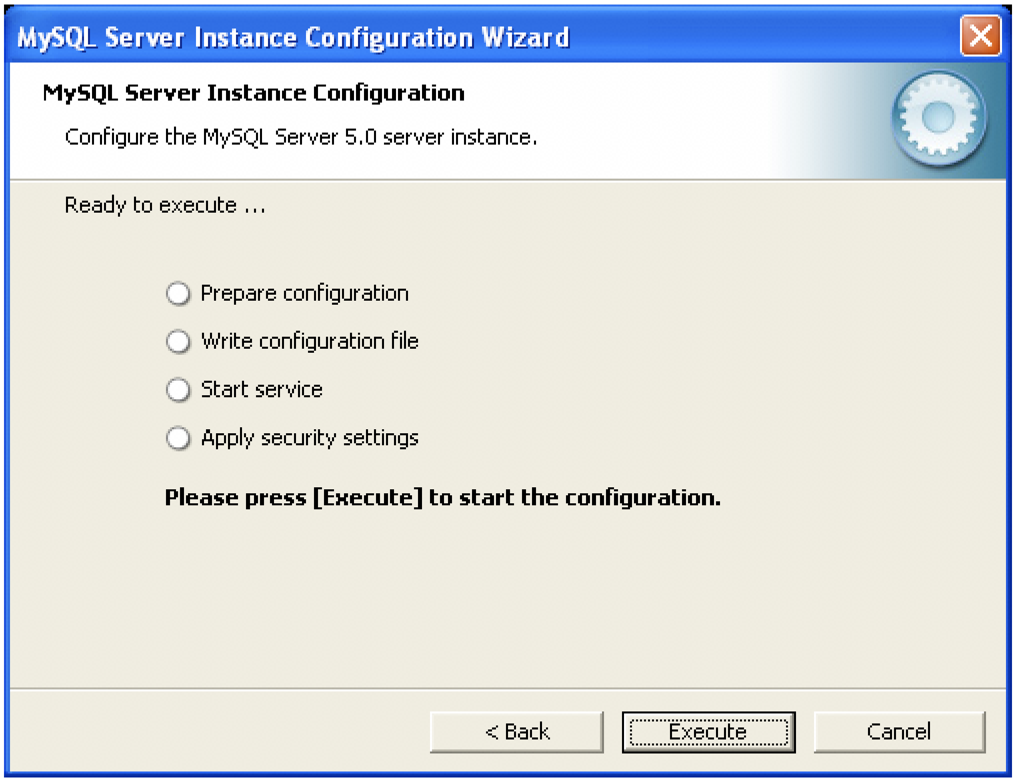
Confirm: Retype the password.

[ ] Enable root access from remote machines

[ ] Create An Anonymous Account

This option will create an anonymous account on this server. Please note that this can lead to an insecure system.

Press Execute:


MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration

Configure the MySQL Server 5.0 server instance.

Ready to execute ...

Prepare configuration

Write configuration file

Start service

Apply security settings

Please press [Execute] to start the configuration.

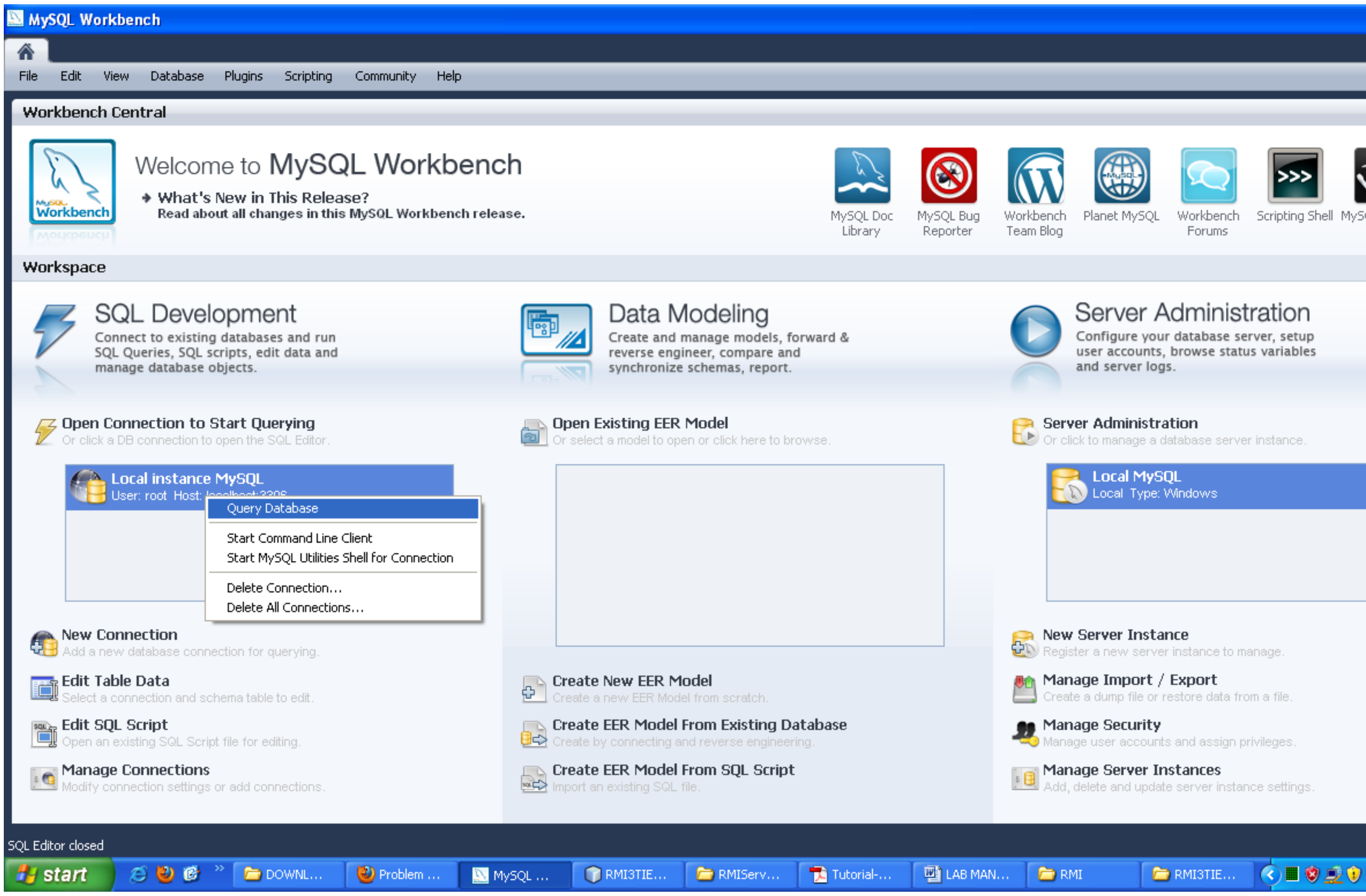
Restart the computer and the installation should be complete.

Install the MySQL Workbench.

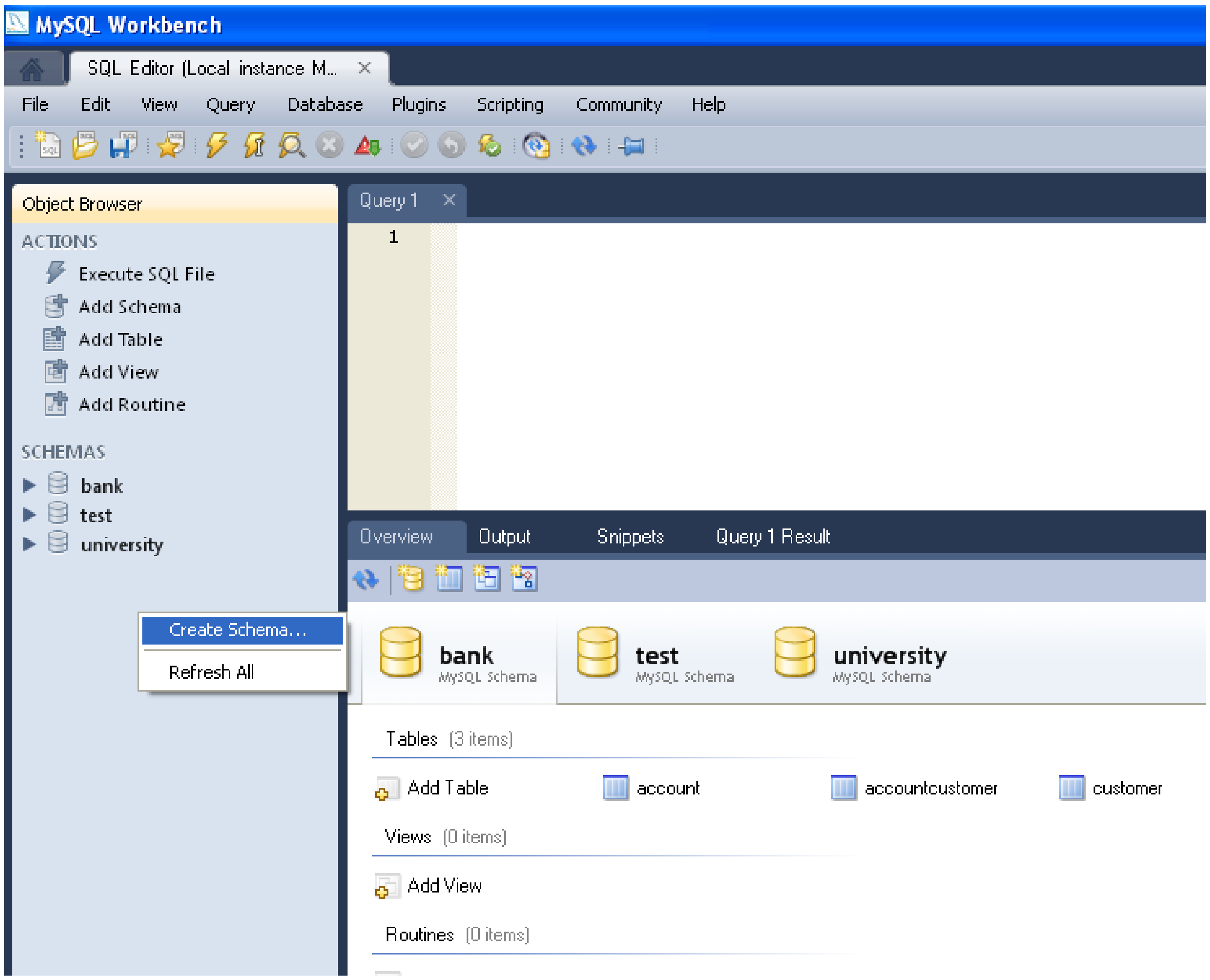
You can create the database in two ways:

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

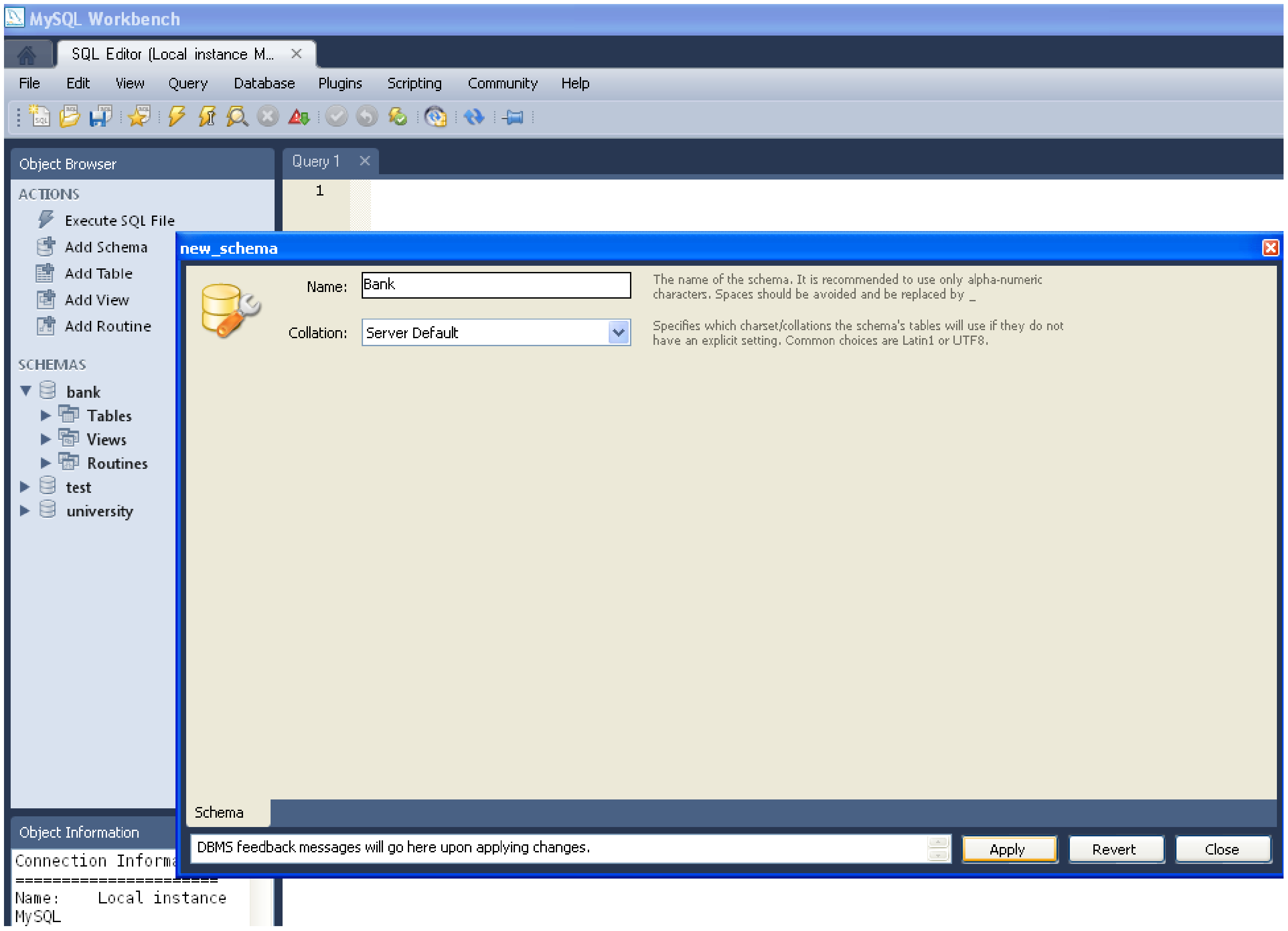
Click on local instance with the right and click Query Database.


MySQL Workbench
File Edit View Database Plugins Scripting Community Help
Workbench Central
Welcome to MySQL Workbench
What's New in This Release?
Read about all changes in this MySQL Workbench release.
MySQL Doc Library
MySQL Bug Reporter
Workbench Team Blog
Planet MySQL
Workbench Forums
Scripting Shell
Workspace
SQL Development
Connect to existing databases and run SQL Queries, SQL scripts, edit data and manage database objects.
Data Modeling
Create and manage models, forward & reverse engineer, compare and synchronize schemas, report.
Server Administration
Configure your database server, setup user accounts, browse status variables and server logs.
Open Connection to Start Querying
Or click a DB connection to open the SQL Editor.
Open Existing EER Model
Or select a model to open or click here to browse.
Server Administration
Or click to manage a database server instance.
Local instance MySQL
User: root Host:
Query Database
Start Command Line Client
Start MySQL Utilities Shell for Connection
Delete Connection...
Delete All Connections...
Local MySQL
Local Type: Windows
New Connection
Add a new database connection for querying.
Edit Table Data
Select a connection and schema table to edit.
Edit SQL Script
Open an existing SQL Script file for editing.
Manage Connections
Modify connection settings or add connections.
Create New EER Model
Create a new EER Model from scratch.
Create EER Model From Existing Database
Create by connecting and reverse engineering.
Create EER Model From SQL Script
Import an existing SQL file.
New Server Instance
Register a new server instance to manage.
Manage Import / Export
Create a dump file or restore data from a file.
Manage Security
Manage user accounts and assign privileges.
Manage Server Instances
Add, delete and update server instance settings.
SQL Editor closed
start

Click with the right and select create schema.



Give a name to the database: and press Apply.



Name: Bank

The name of the schema. It is recommended to use only alpha-numeric characters. Spaces should be avoided and be replaced by _

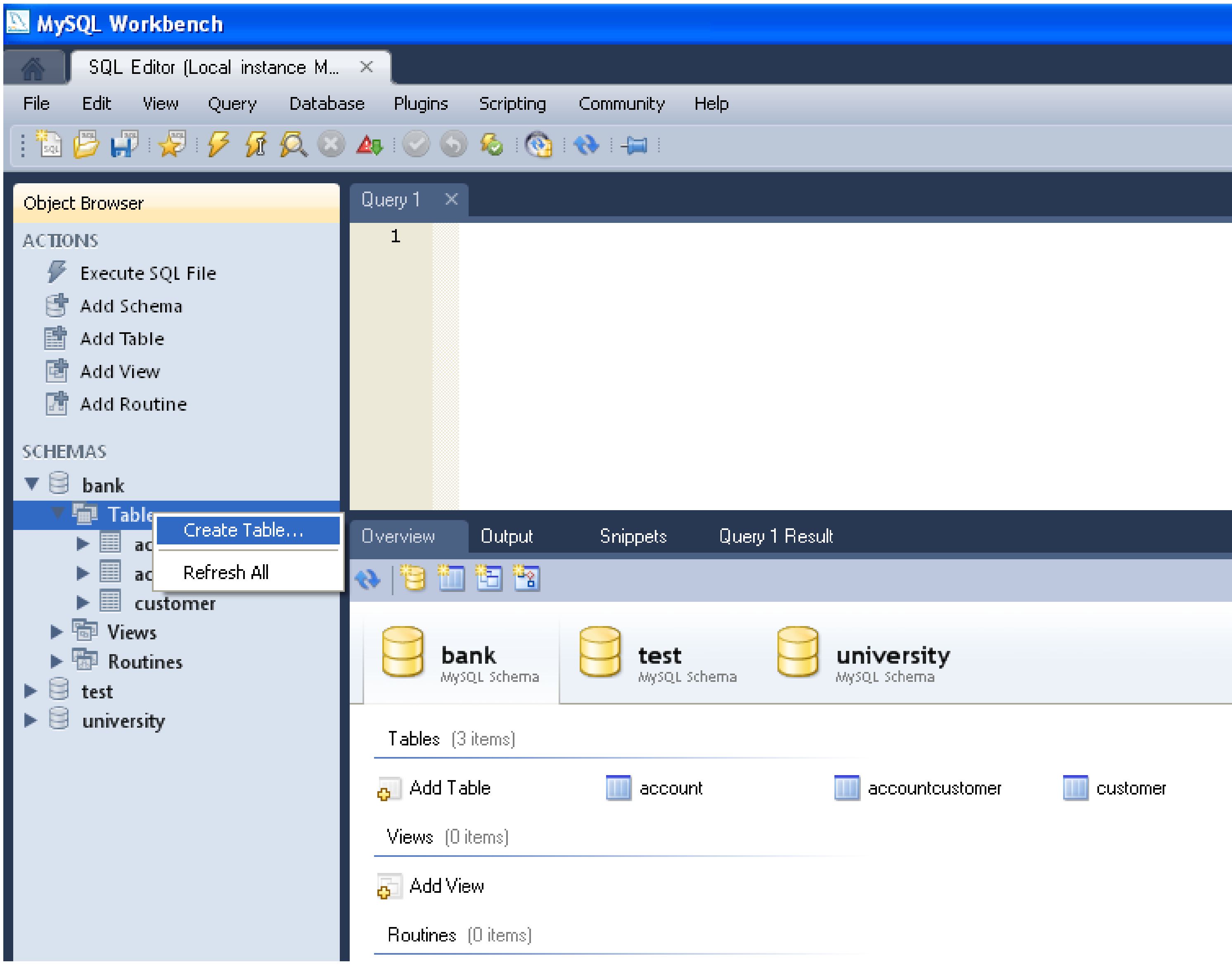
Collation: Server Default

Specifies which charset/collations the schema's tables will use if they do not have an explicit setting. Common choices are Latin1 or UTF8.

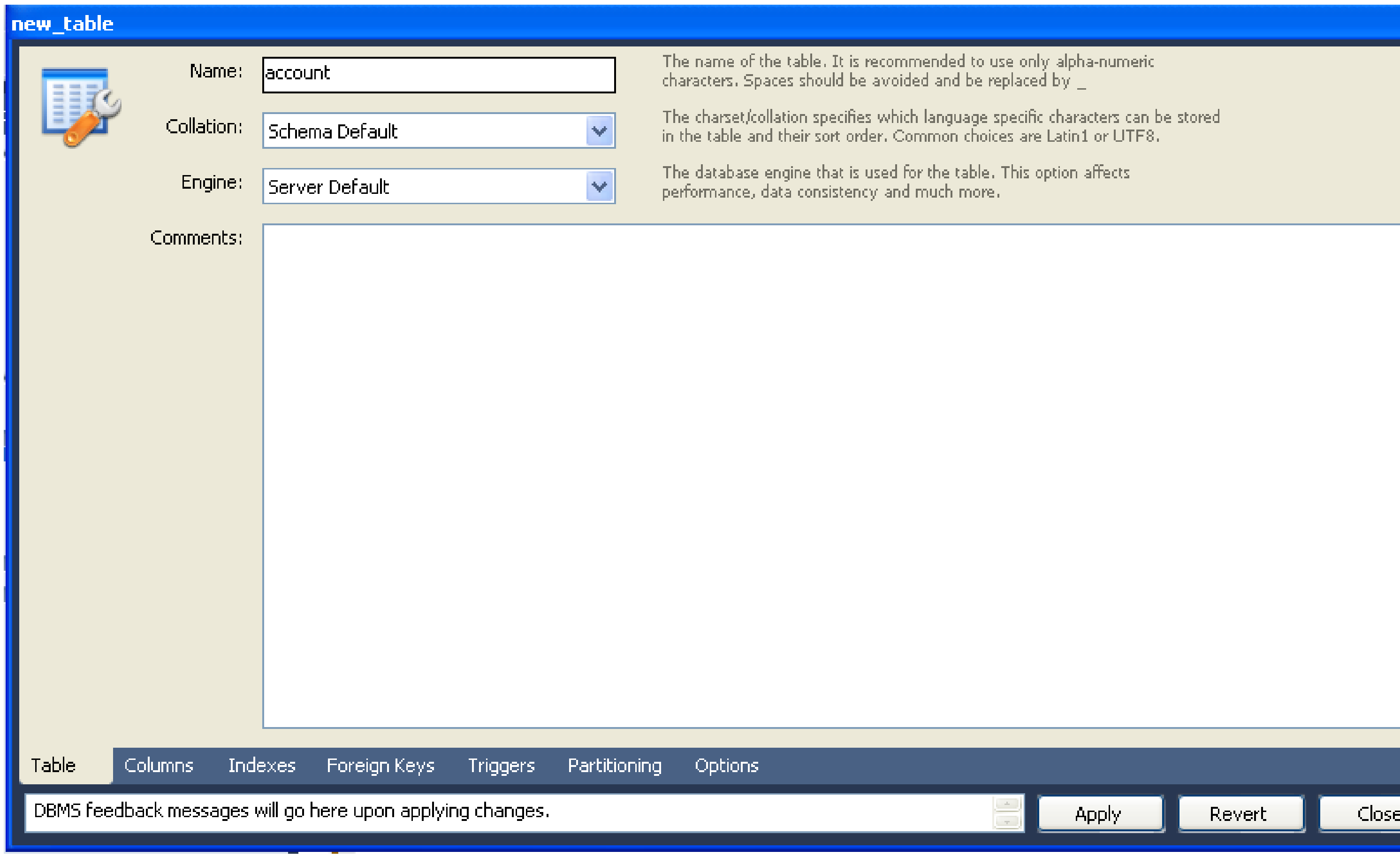
DBMS feedback messages will go here upon applying changes.

```
Connection Informa
======================
Name:     Local instance
MySQL
```

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



Choose a name for the Table:



Name: account

The name of the table. It is recommended to use only alpha-numeric characters. Spaces should be avoided and be replaced by _

Collation: Schema Default

The charset/collation specifies which language specific characters can be stored in the table and their sort order. Common choices are Latin1 or UTF8.

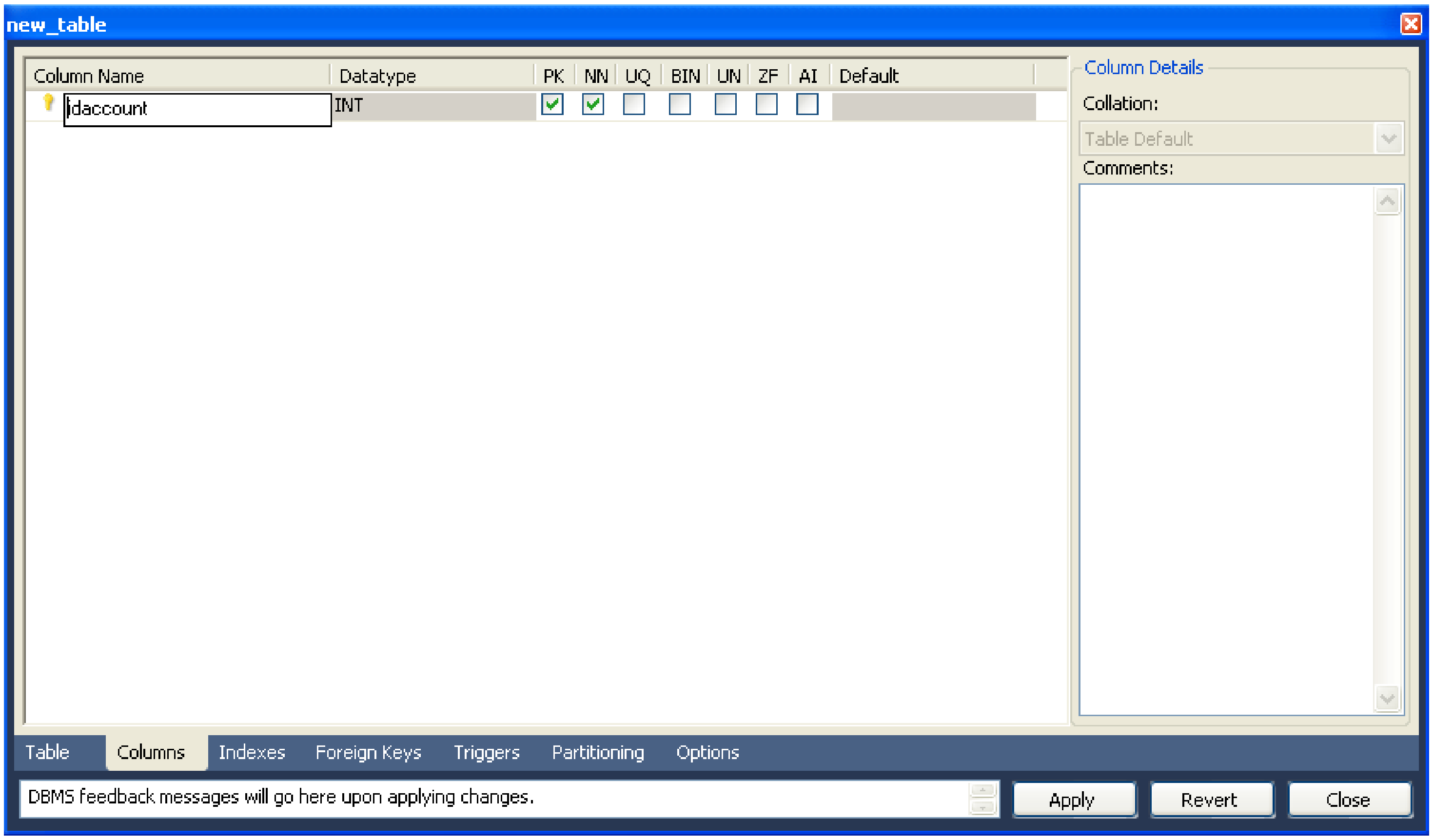
Engine: Server Default

The database engine that is used for the table. This option affects performance, data consistency and much more.

Comments:

DBMS feedback messages will go here upon applying changes.

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



Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
idaccount	INT	✓	✓						

Column Details

Collation: Table Default

Comments:

DBMS feedback messages will go here upon applying changes.

Following the above procedure create three tables:

Table Account
Fields: IdAccount (int), Balance (float)

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

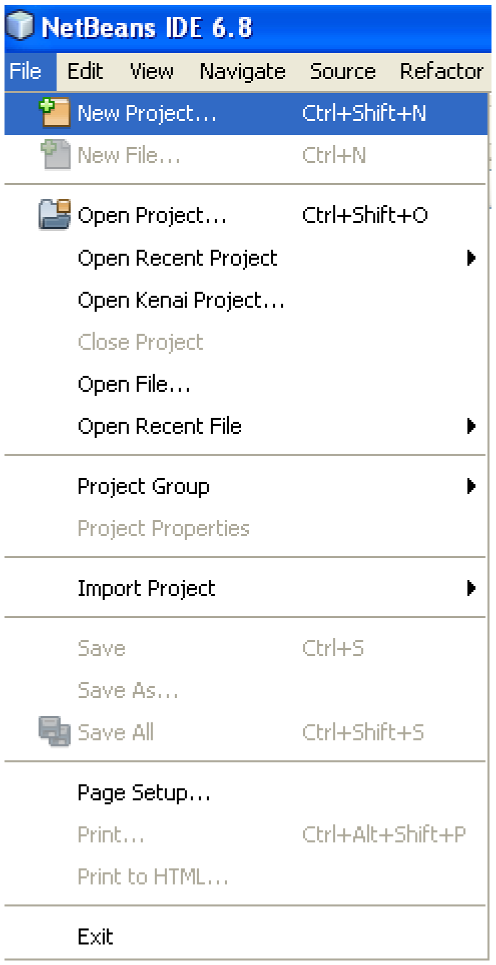
Table AccountCustomer
Fields: idAccount, IdCustomer

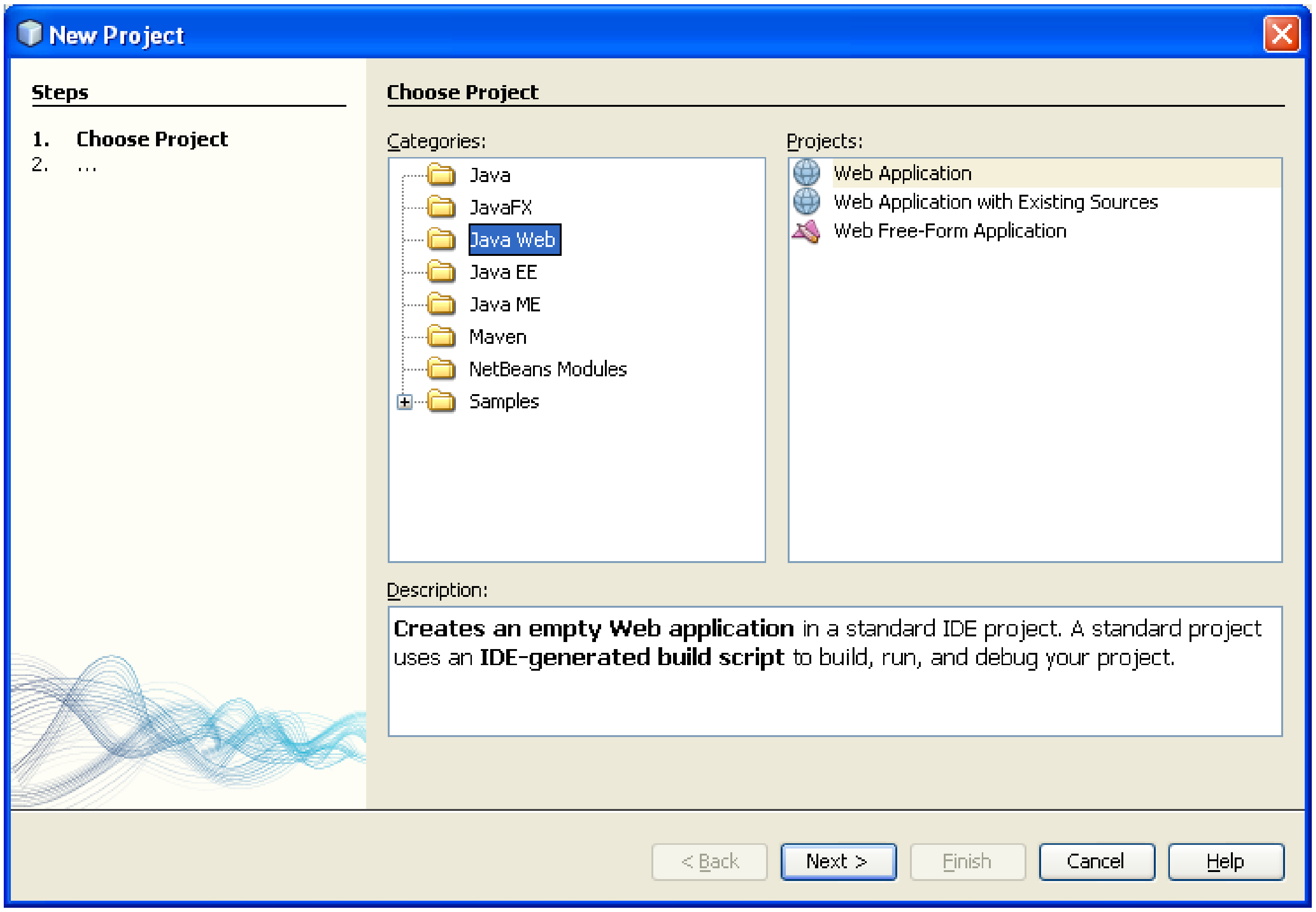
2. Developing a web banking application

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

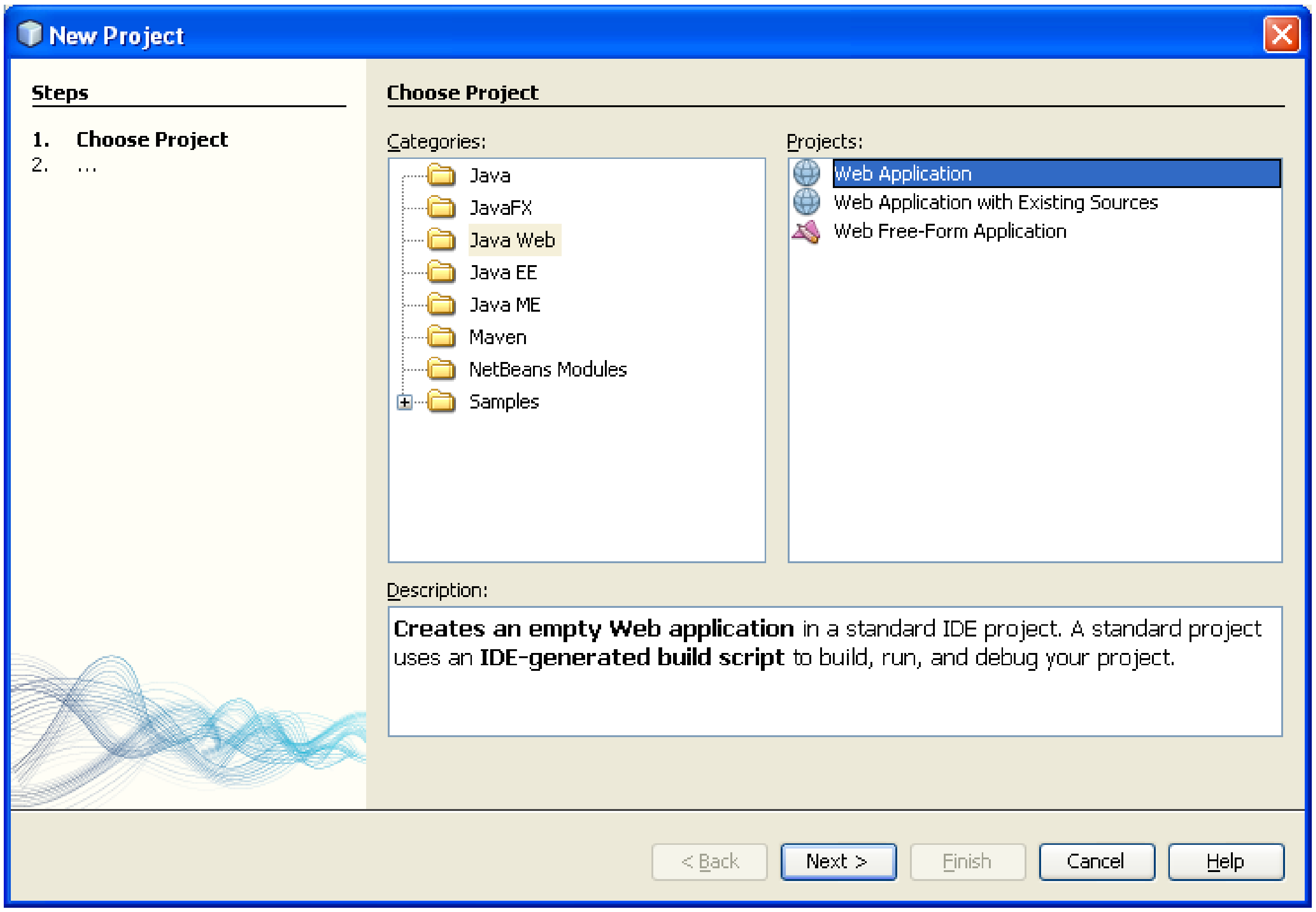
Perform the following in order:

Create the project:

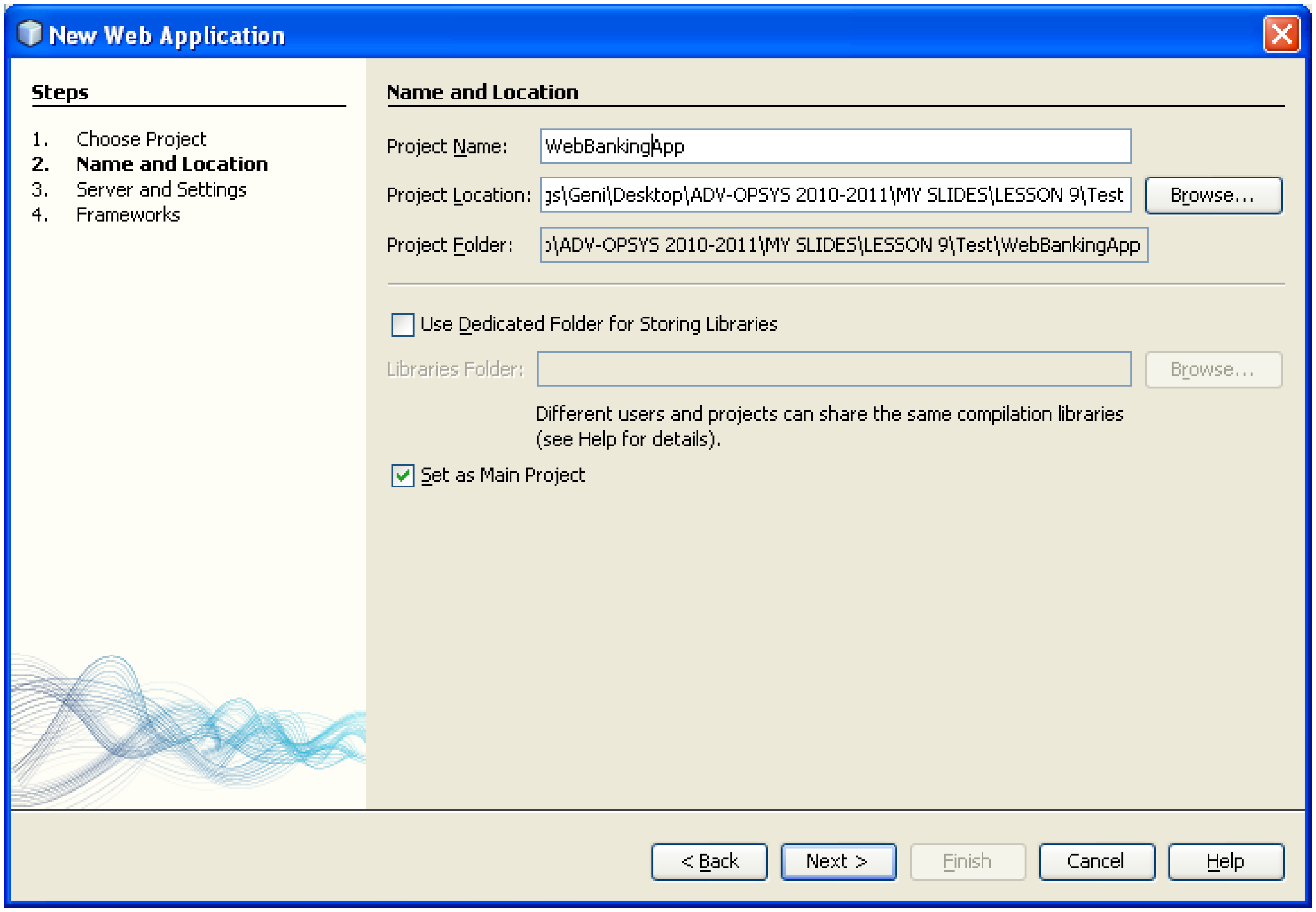

NetBeans IDE 6.8
File
Edit
View
Navigate
Source
Refactor
New Project... Ctrl+Shift+N
New File... Ctrl+N
Open Project... Ctrl+Shift+O
Open Recent Project
Open Kenai Project...
Close Project
Open File...
Open Recent File
Project Group
Project Properties
Import Project
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Page Setup...
Print... Ctrl+Alt+Shift+P
Print to HTML...
Exit



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.



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.



Project Name: WebBankingApp

Project Location: gs\Geni\Desktop\ADV-OPSYS 2010-2011\MY SLIDES\LESSON 9\Test

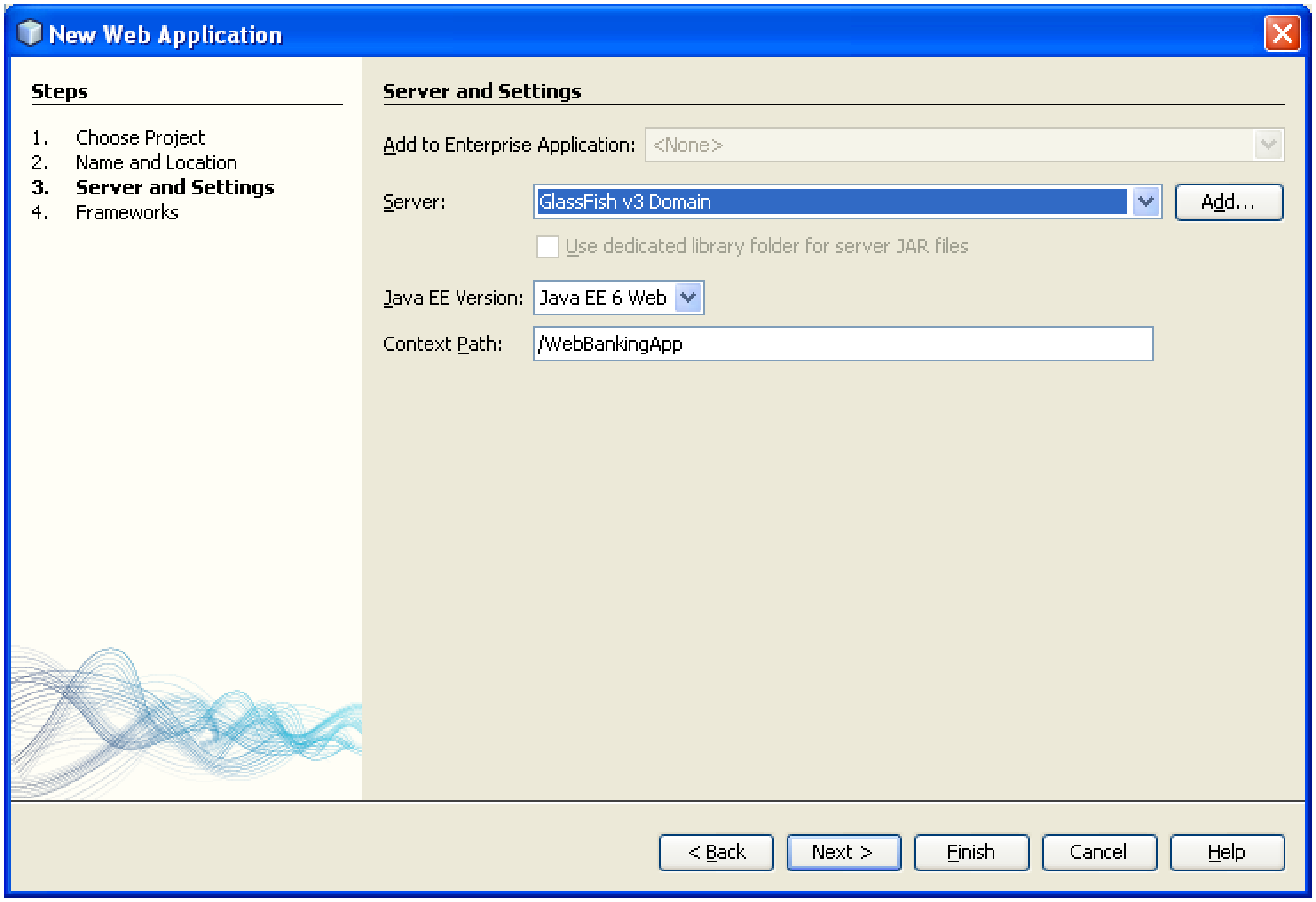
Project Folder: p\ADV-OPSYS 2010-2011\MY SLIDES\LESSON 9\Test\WebBankingApp

[ ] 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

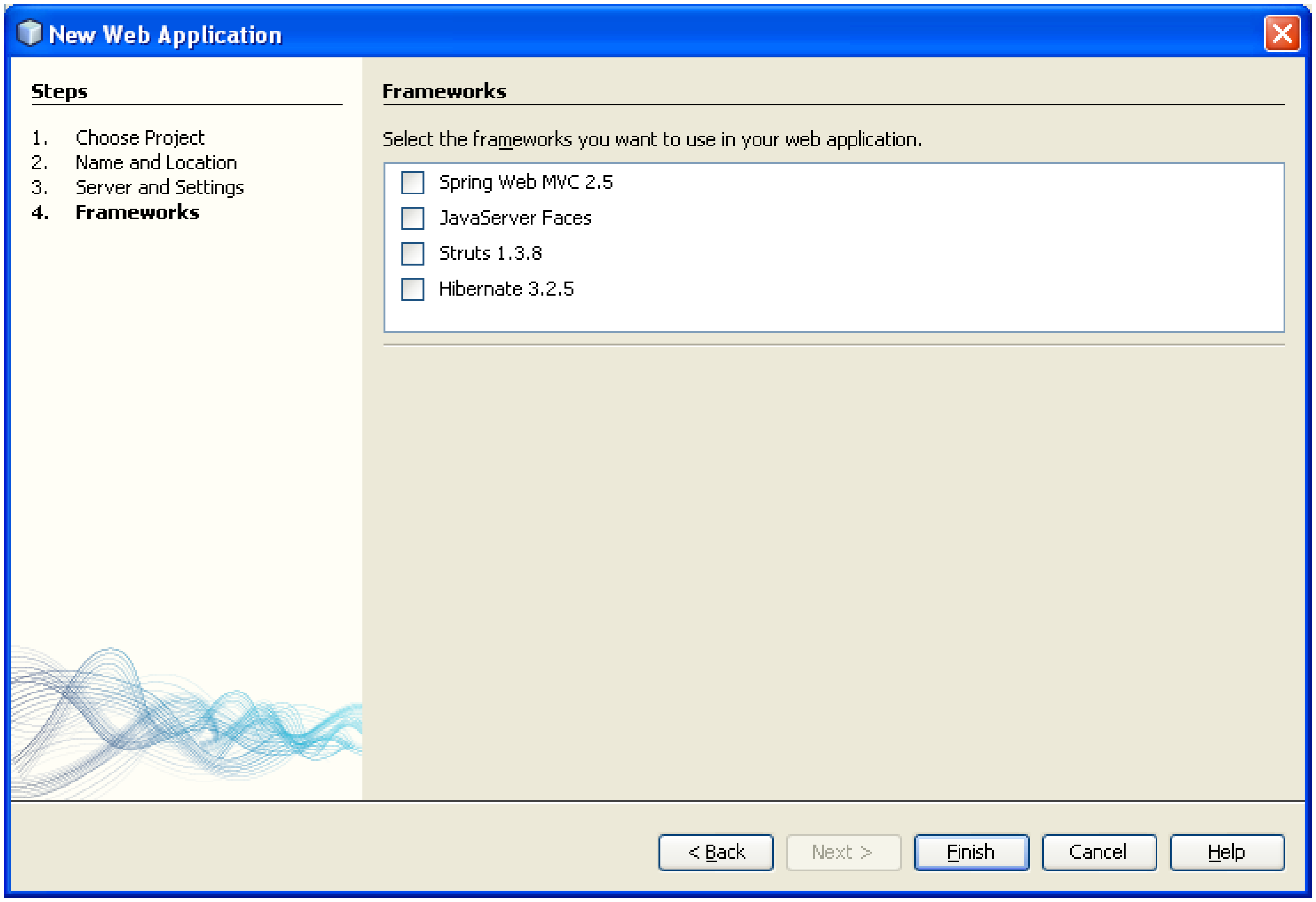
Add to Enterprise Application: <None>

Server: GlassFish v3 Domain

Use dedicated library folder for server JAR files

Java EE Version: Java EE 6 Web

Context Path: /WebBankingApp


New Web Application

Steps

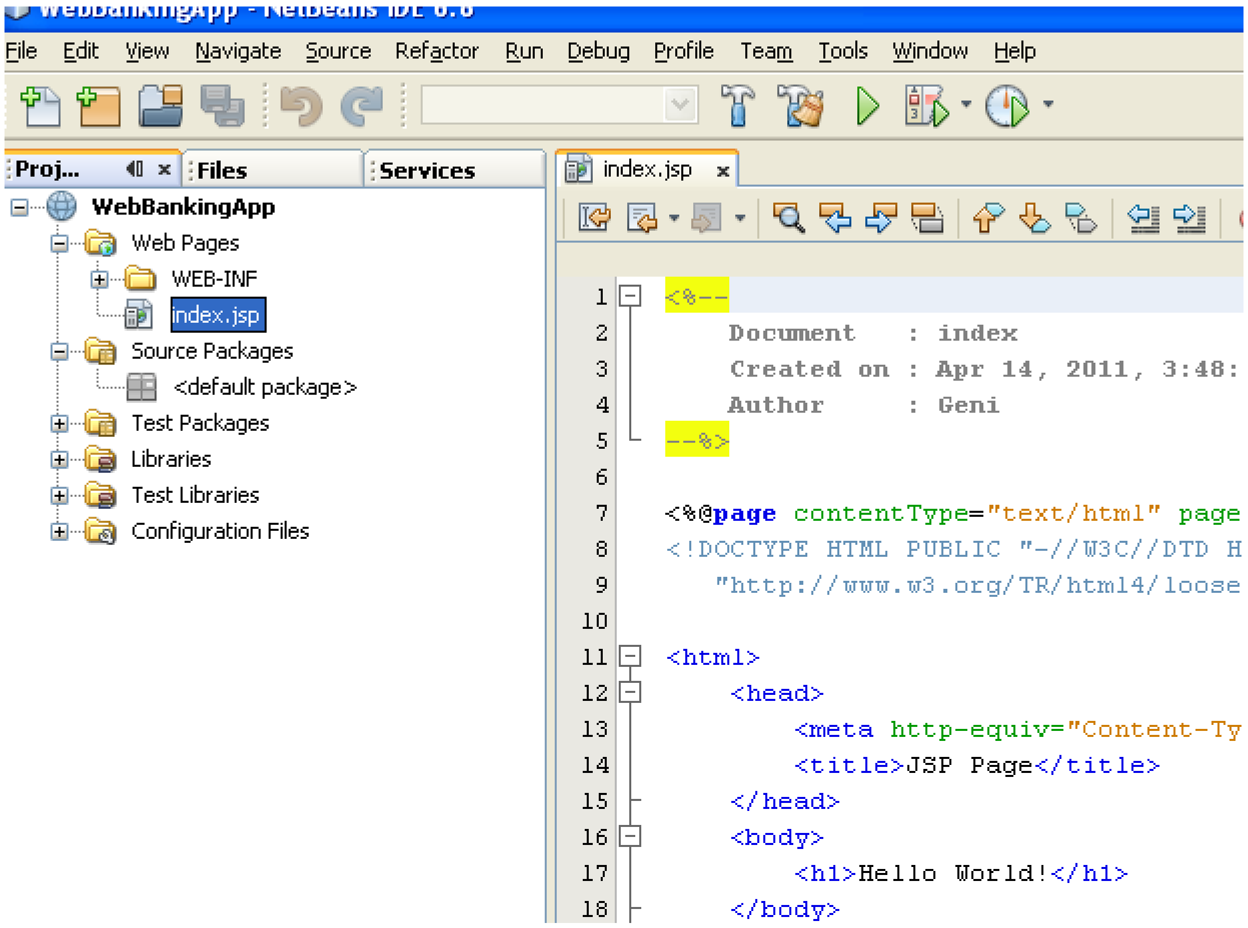
1. Choose Project
2. Name and Location
3. Server and Settings
4. **Frameworks**

Frameworks

Select the frameworks you want to use in your web application.

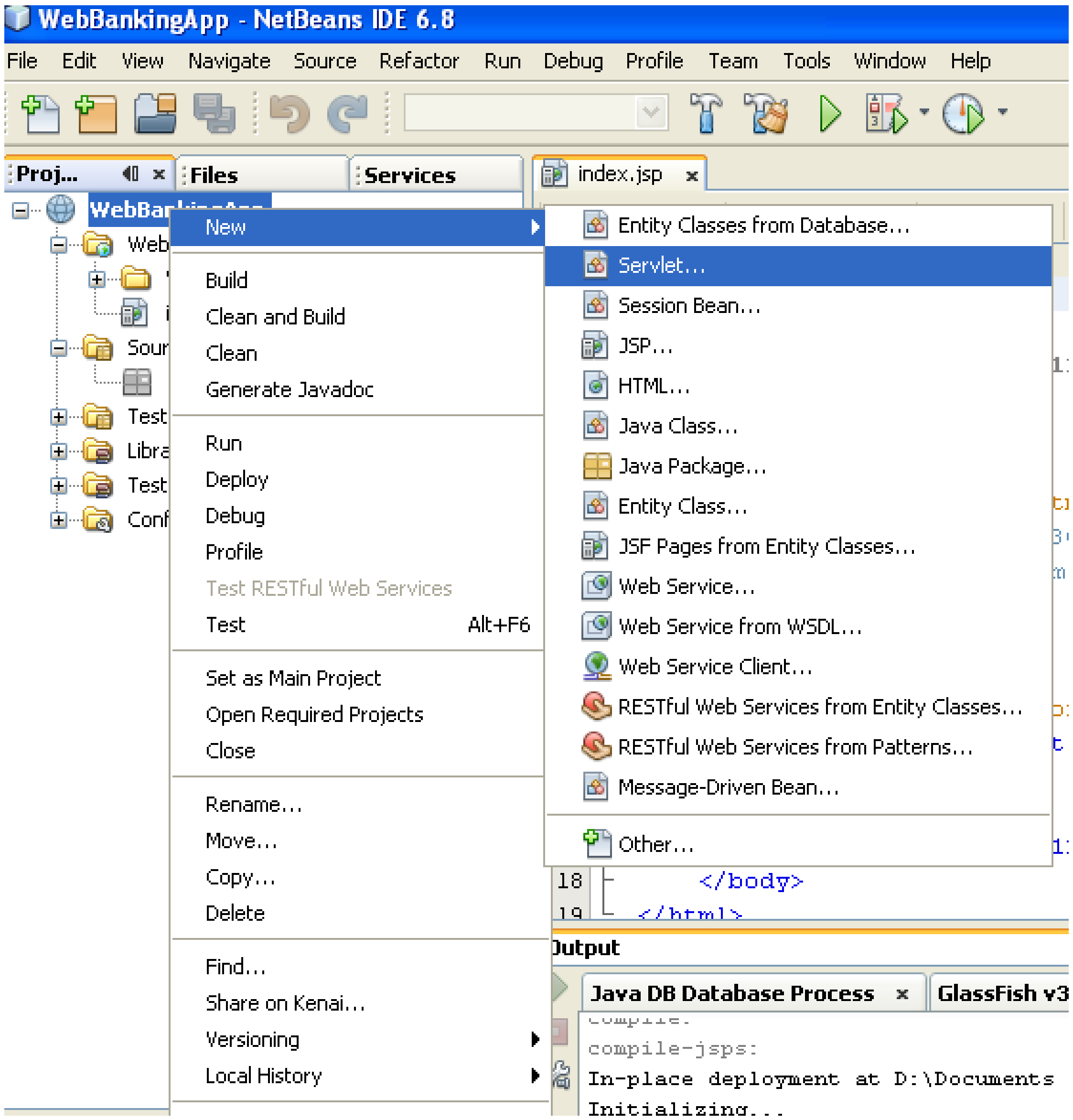
- [ ] Spring Web MVC 2.5
- [ ] JavaServer Faces
- [ ] Struts 1.3.8
- [ ] Hibernate 3.2.5

The application created is this one:



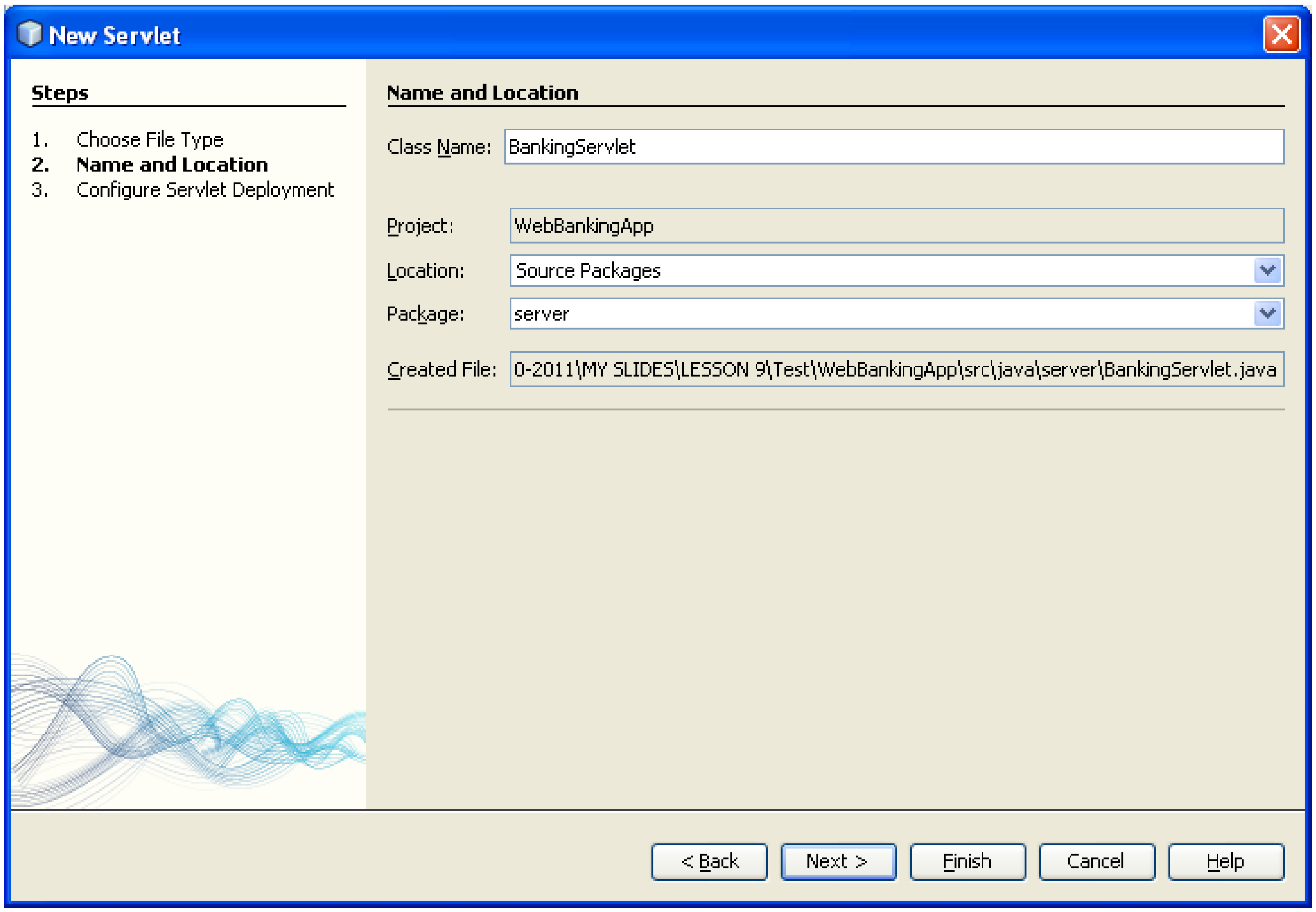
```
<%--
    Document   : index
    Created on : Apr 14, 2011, 3:48:
    Author     : Geni
--%>

<%@page contentType="text/html" page
<!DOCTYPE HTML PUBLIC "-//W3C//DTD H
   "http://www.w3.org/TR/html4/loose

<html>
    <head>
        <meta http-equiv="Content-Ty
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>
    </body>
```

Create a Servlet:



```
        </body>
    </html>
```

```
compile-jsps:
In-place deployment at D:\Documents
Initializing...
```



Steps

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

Name and Location

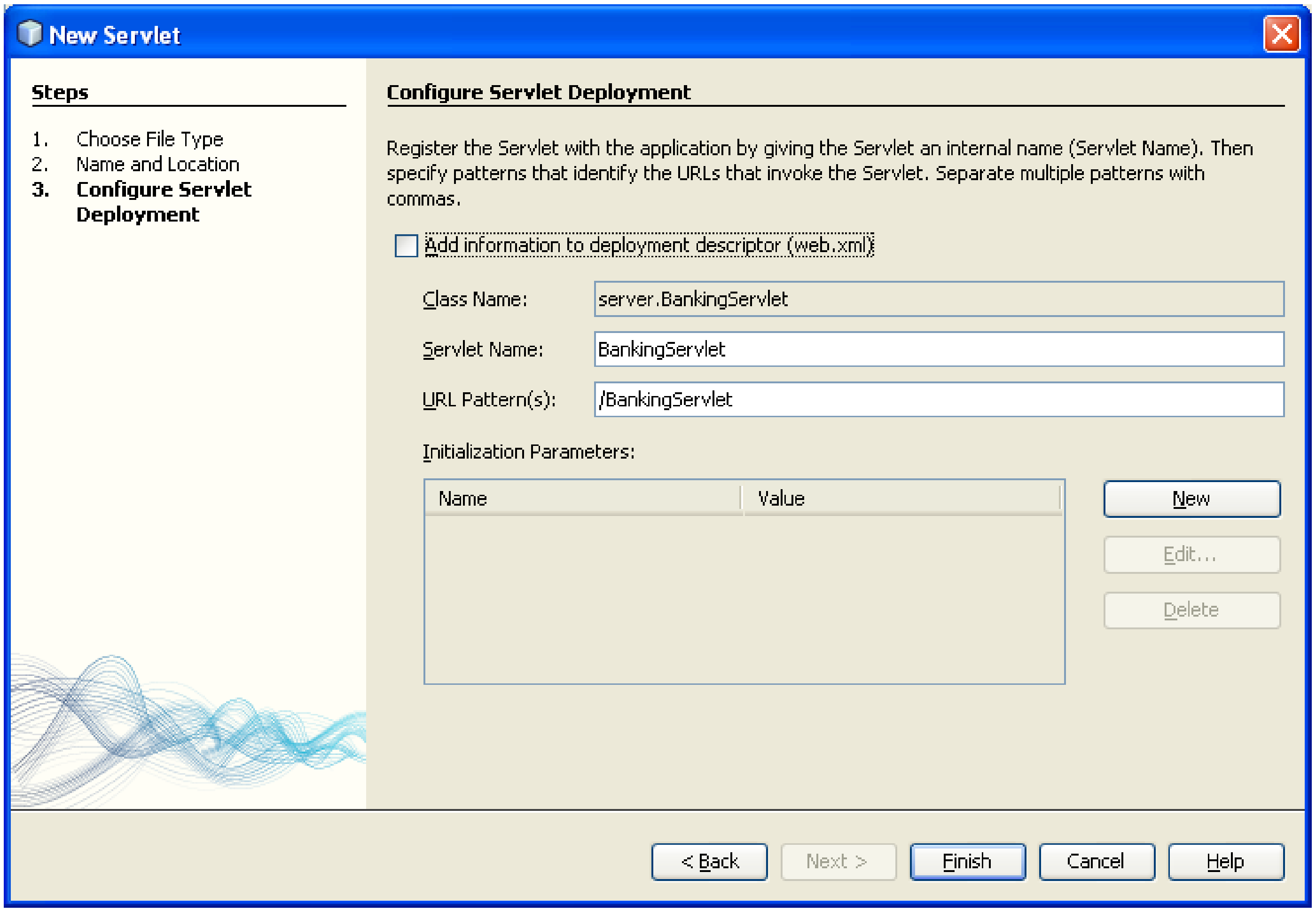
Class Name: BankingServlet

Project: WebBankingApp

Location: Source Packages

Package: server

Created File: 0-2011\MY SLIDES\LESSON 9\Test\WebBankingApp\src\java\server\BankingServlet.java


New Servlet

Steps

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

Configure Servlet Deployment

Register the Servlet with the application by giving the Servlet an internal name (Servlet Name). Then specify patterns that identify the URLs that invoke the Servlet. Separate multiple patterns with commas.

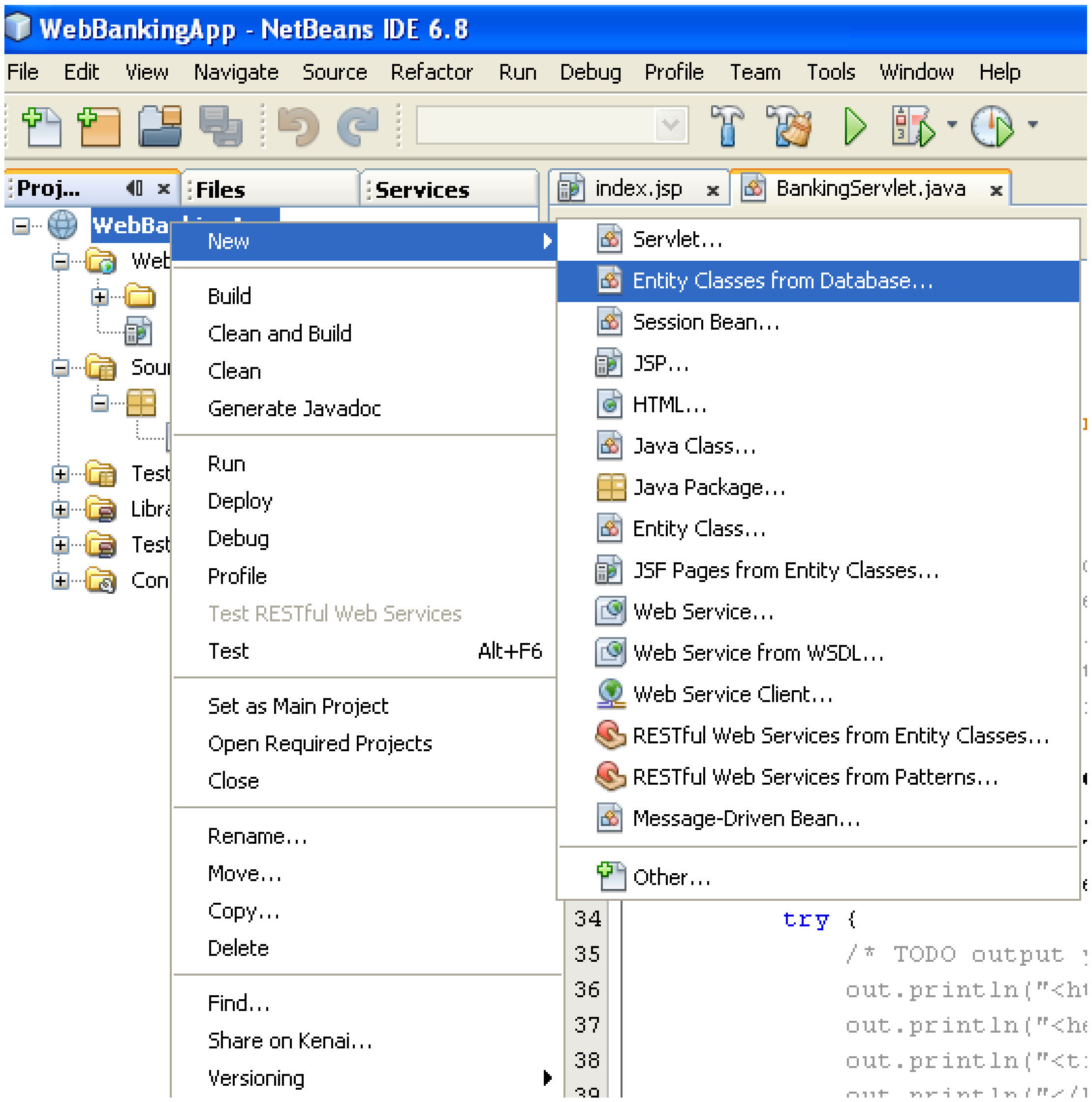
☐ Add information to deployment descriptor (web.xml)

Class Name:	server.BankingServlet
Servlet Name:	BankingServlet
URL Pattern(s):	/BankingServlet

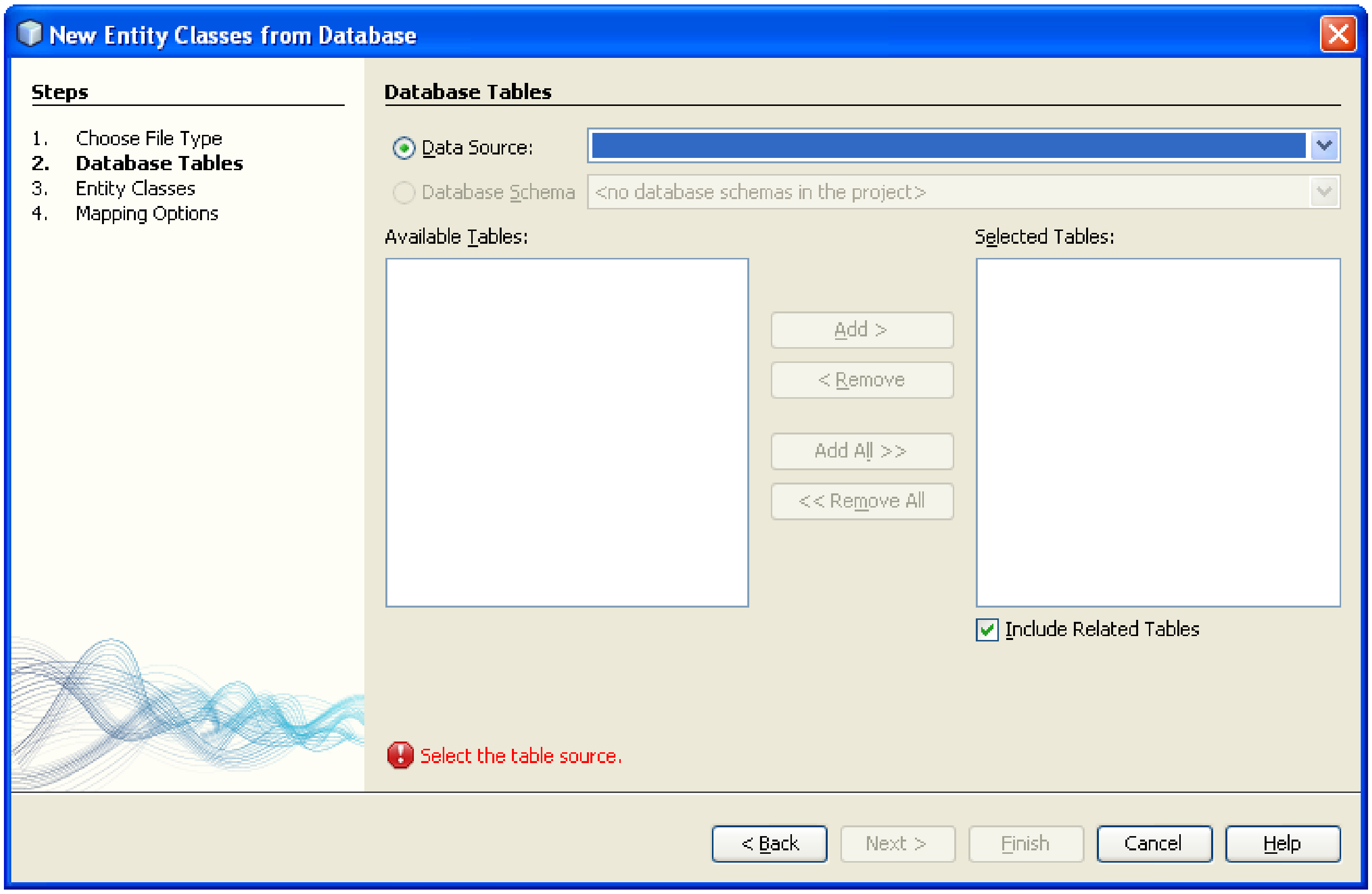
Initialization Parameters:

Name	Value

Create an EJB to connect to the database MySQL:



```
            try {
                /* TODO output y
                out.println("<ht
                out.println("<he
                out.println("<t:
```



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:

Add >

< Remove

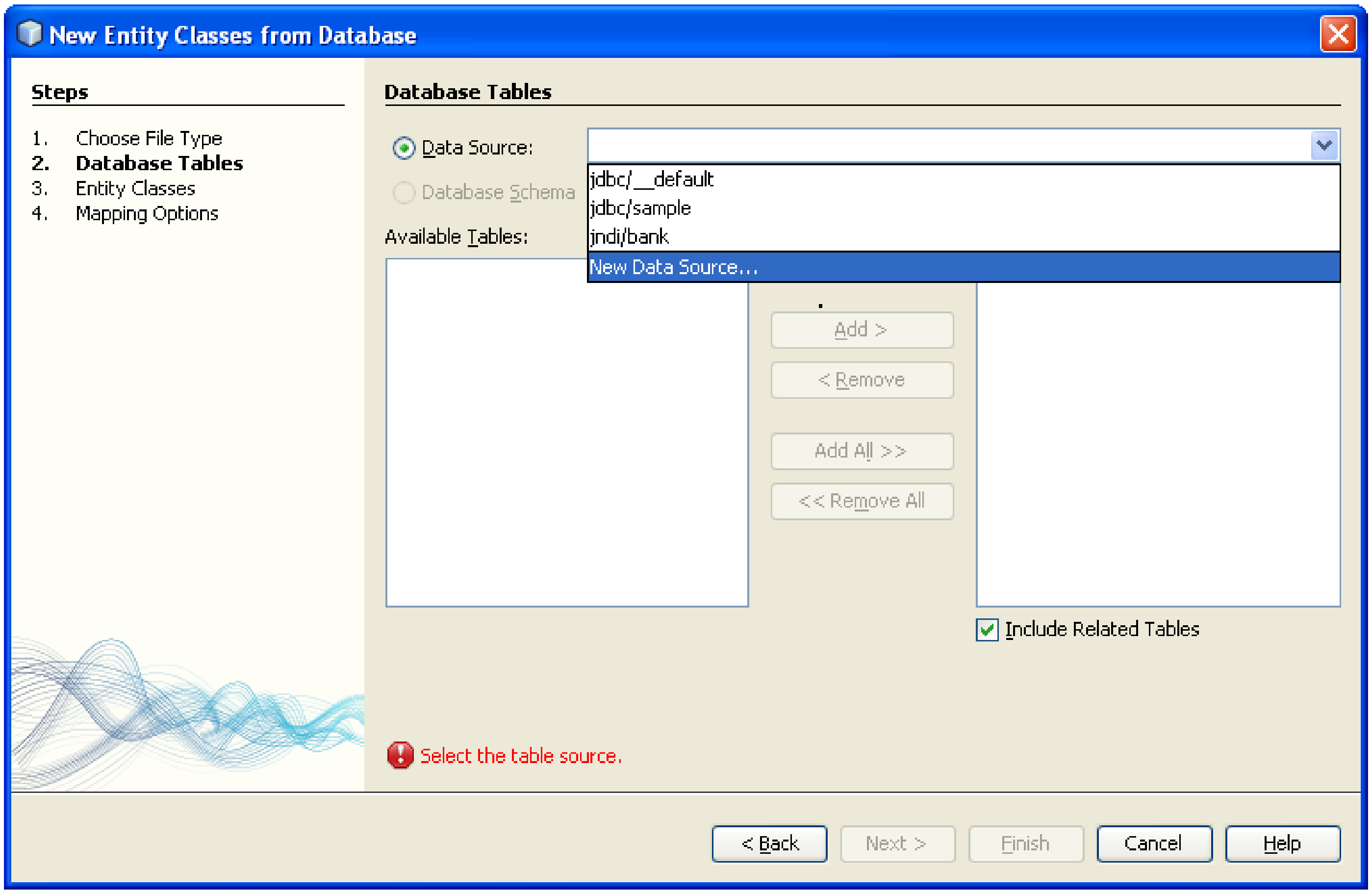
Add All >>

<< Remove All

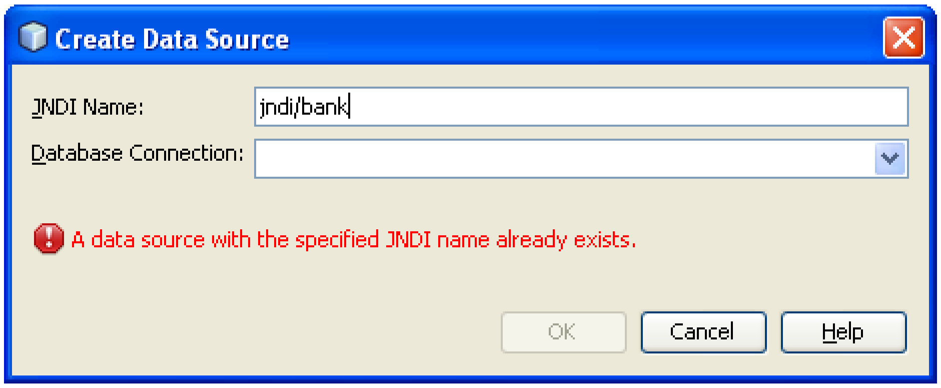
✓ Include Related Tables

Select the table source.

< Back | Next > | Finish | Cancel | Help



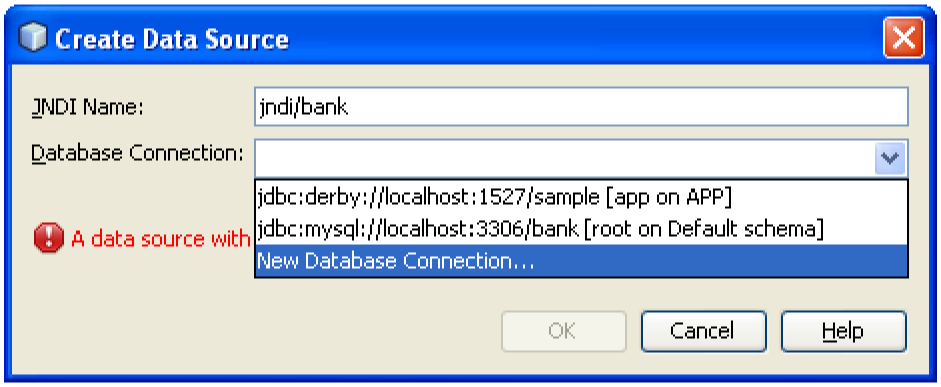
Select the table source.



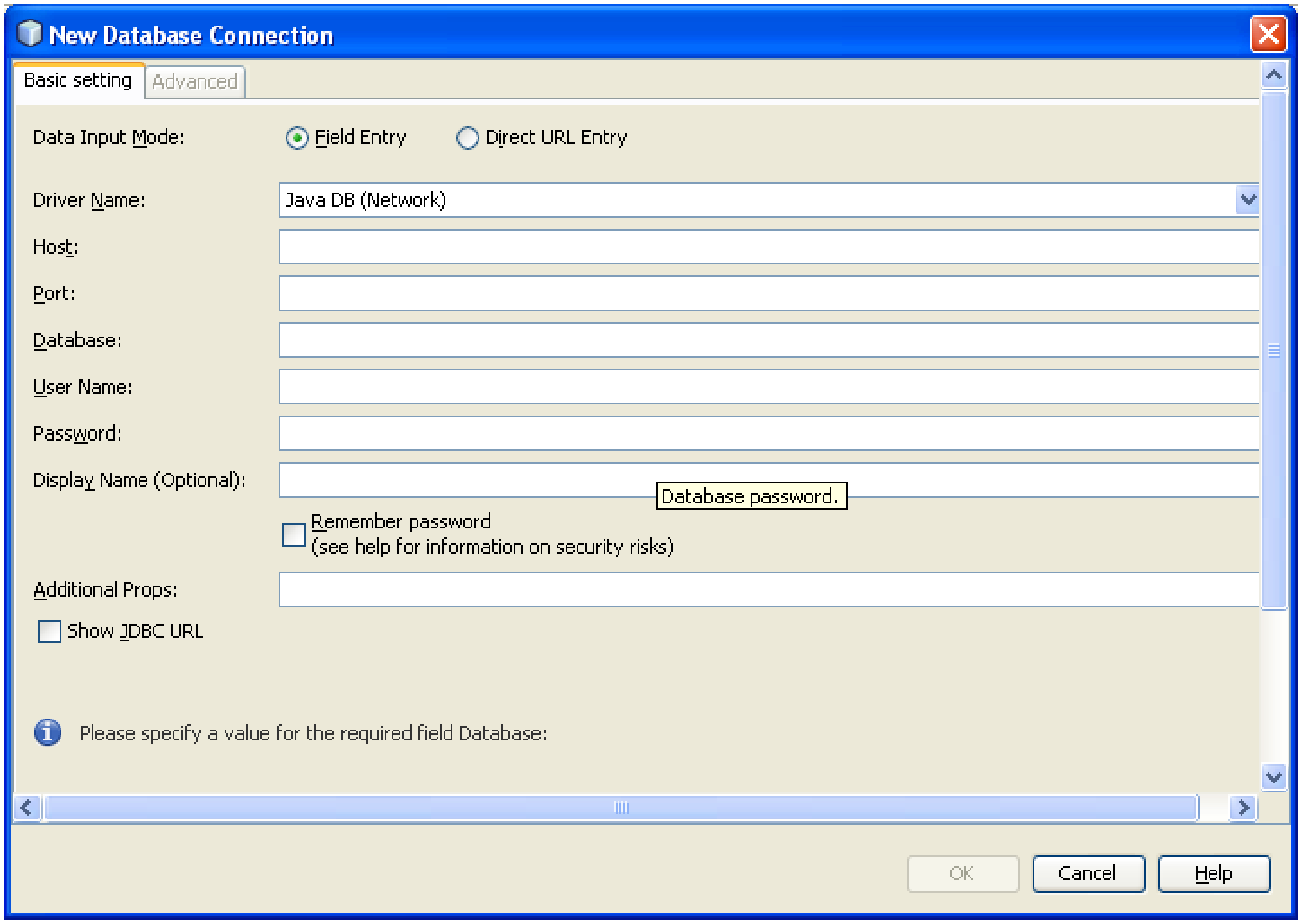
JNDI Name: jndi/bank

A data source with the specified JNDI name already exists.

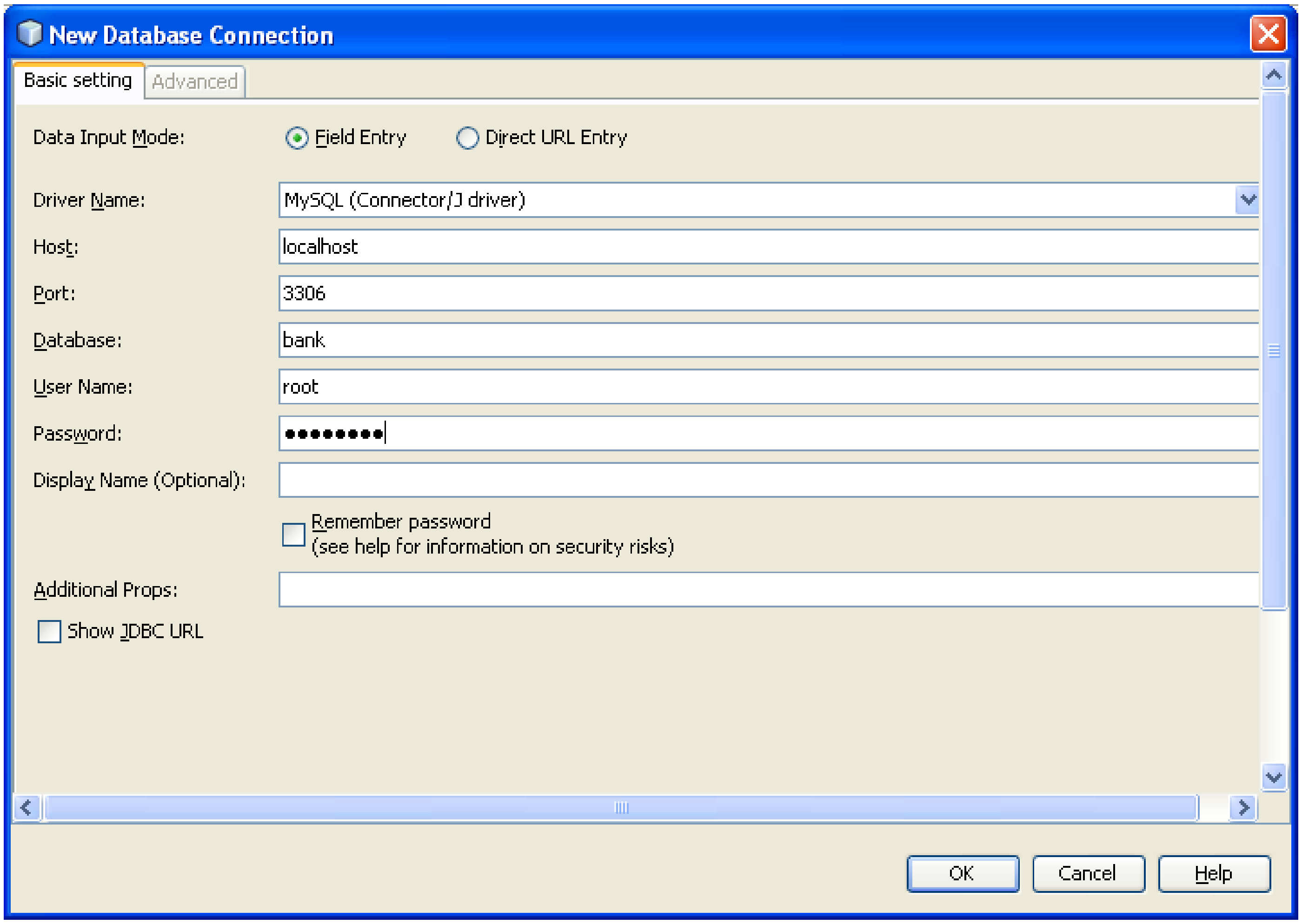
If you do not have the connection to MySQL select “New Database Connection” as follows:


Create Data Source
JNDI Name: jndi/bank
Database Connection:
jdbc:derby://localhost:1527/sample [app on APP]
jdbc:mysql://localhost:3306/bank [root on Default schema]
New Database Connection...
A data source with
OK
Cancel
Help

The following page appears:



Fill in the fields as follows;



Data Input Mode: [✓ Field Entry] Direct URL Entry

Driver Name: MySQL (Connector/J driver)

Host: localhost

Port: 3306

Database: bank

User Name: root

Password: ●●●●●●●●●

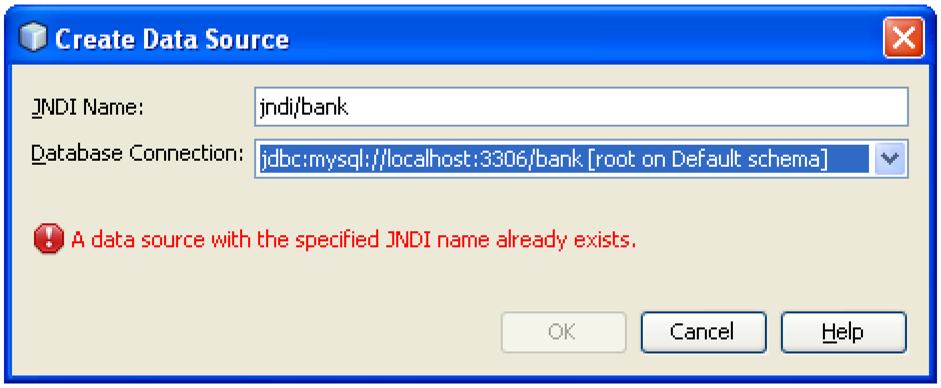
Display Name (Optional):

Remember password (see help for information on security risks)

Additional Props:

Show JDBC URL

When you press "OK" the following page appears again:

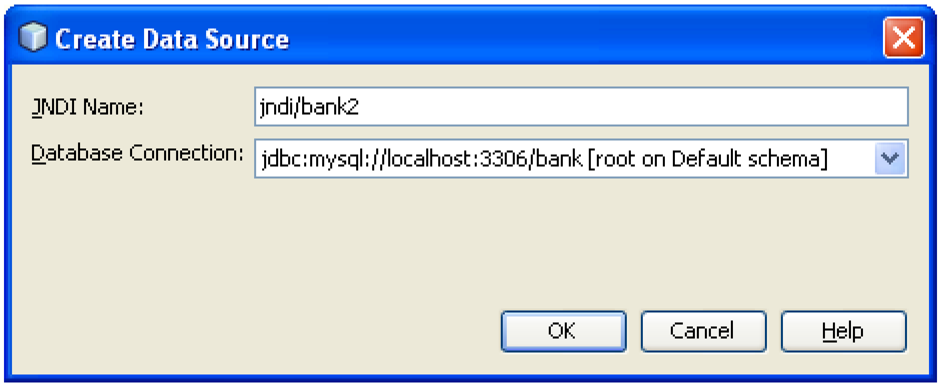


JNDI Name: jndi/bank

Database Connection: jdbc:mysql://localhost:3306/bank [root on Default schema]

A data source with the specified JNDI name already exists.

(In your case the "jndi/bank" does not exist. I show below the illustration for a connection with another name:

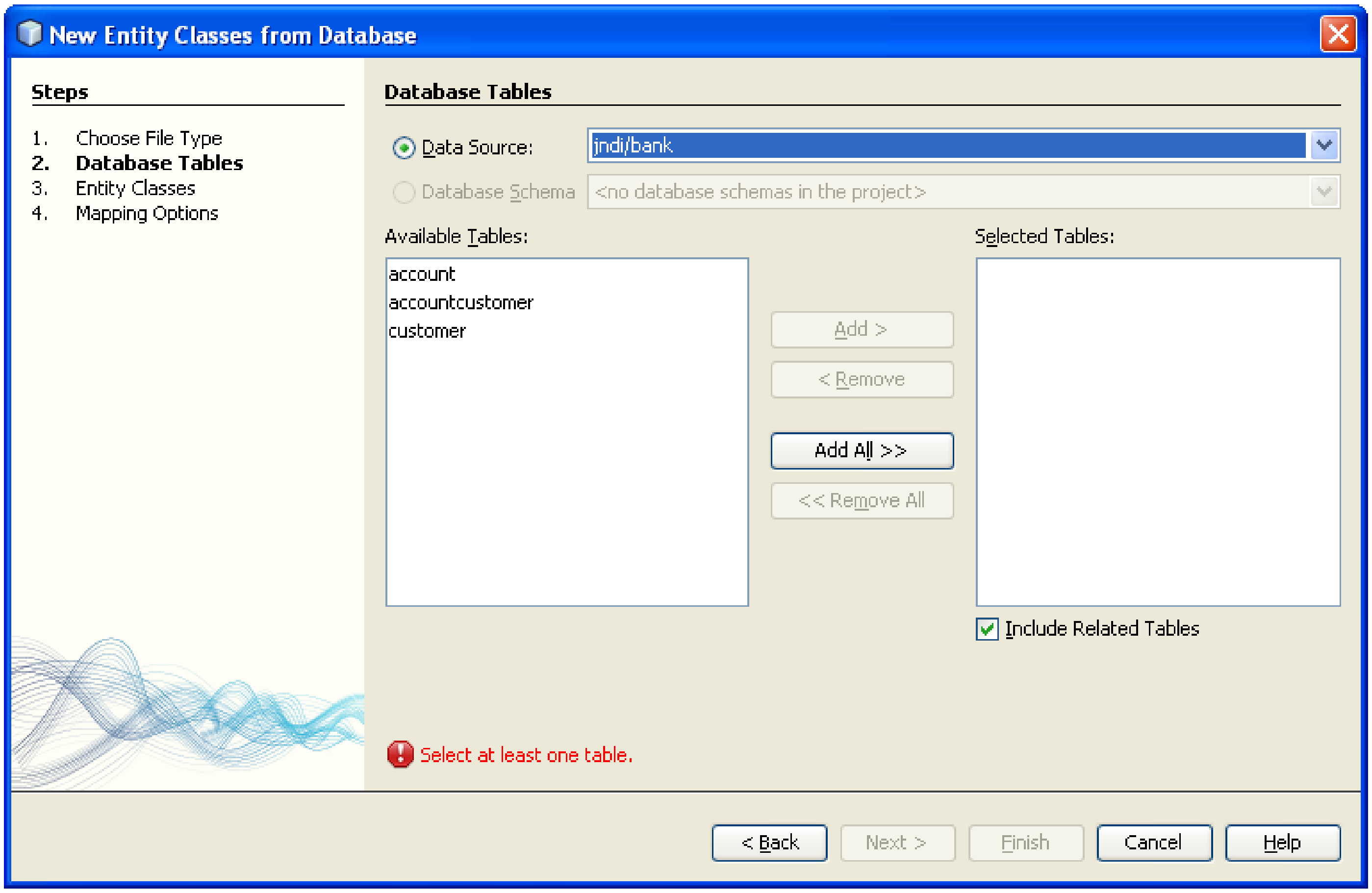


JNDI Name: jndi/bank2

Database Connection: jdbc:mysql://localhost:3306/bank [root on Default schema]

Press OK and you should have the following page:

Choose



[✓ Data Source:] jndi/bank

Database Schema <no database schemas in the project>

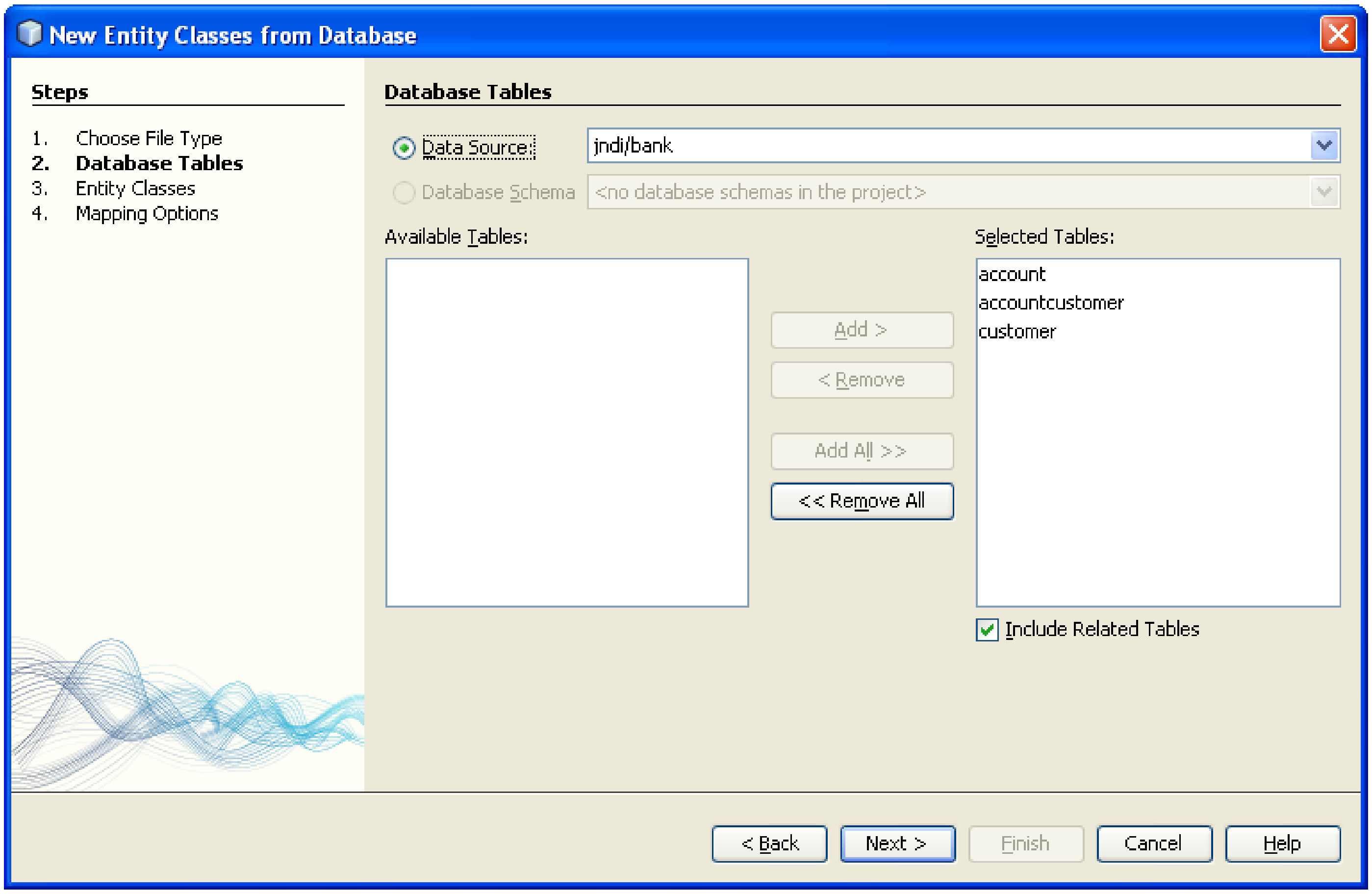
Available Tables:
account
accountcustomer
customer

Selected Tables:

✓ Include Related Tables

Select at least one table.

Now select the tables with “Add All”:



Database Tables

[✓ Data Source:] jndi/bank

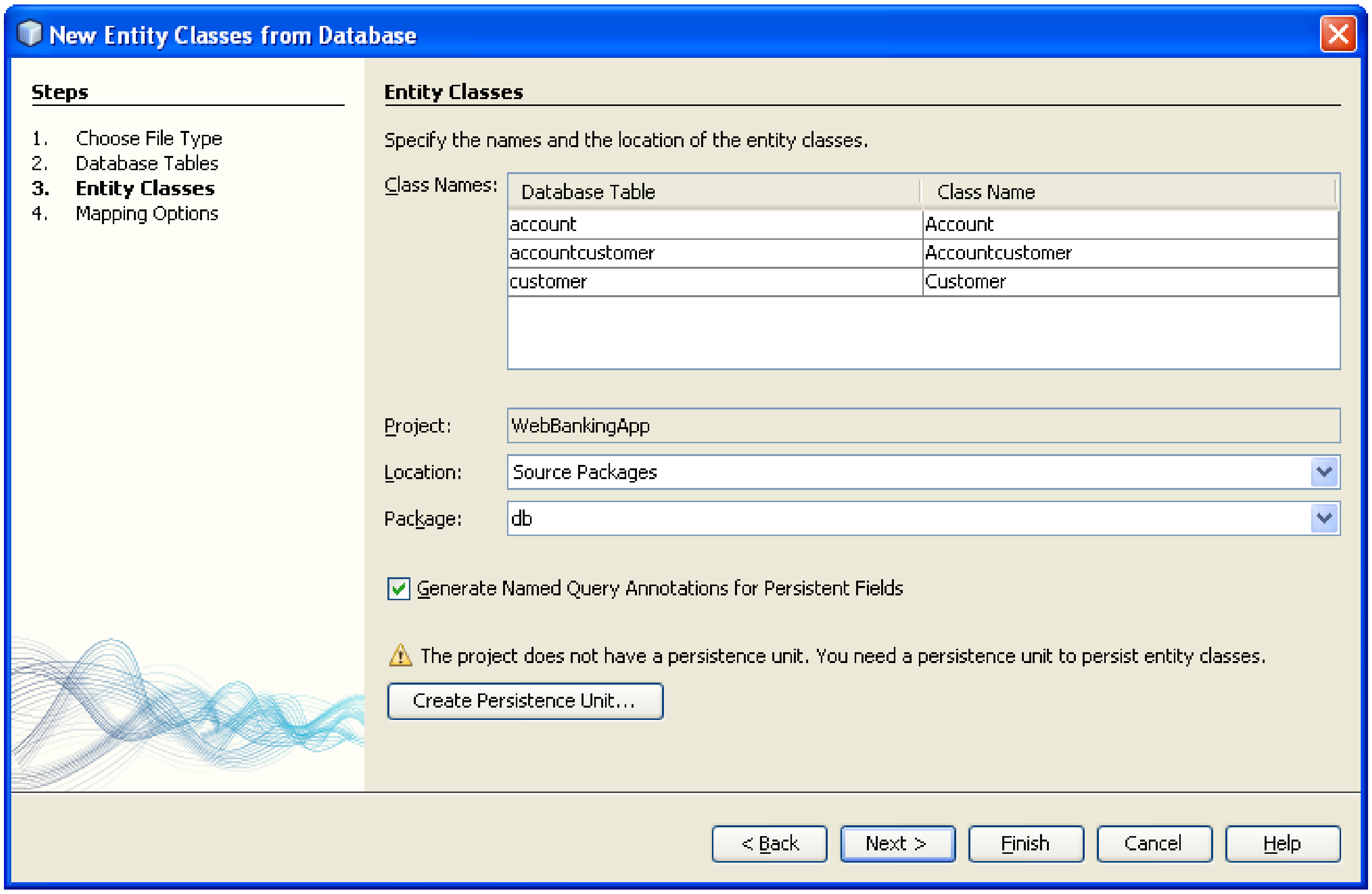
Database Schema <no database schemas in the project>

Available Tables:

Selected Tables:

account
accountcustomer
customer

[✓] Include Related Tables


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
account	Account
accountcustomer	Accountcustomer
customer	Customer

Project: WebBankingApp

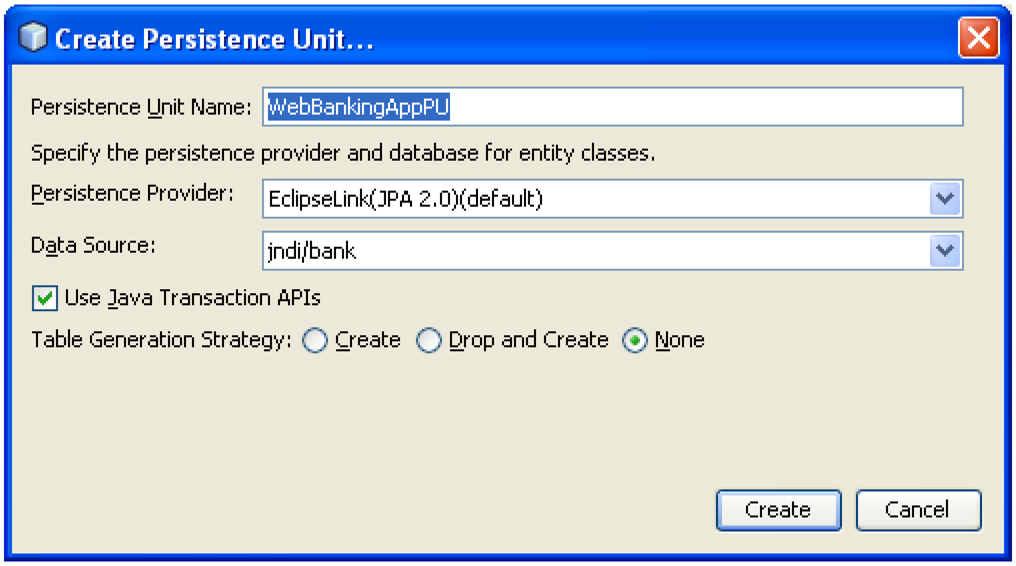
Location: Source Packages

Package: db

[✓] Generate Named Query Annotations for Persistent Fields

⚠ The project does not have a persistence unit. You need a persistence unit to persist entity classes.

Click on “Create Persistence Unit”:


Create Persistence Unit...

Persistence Unit Name: WebBankingAppPU

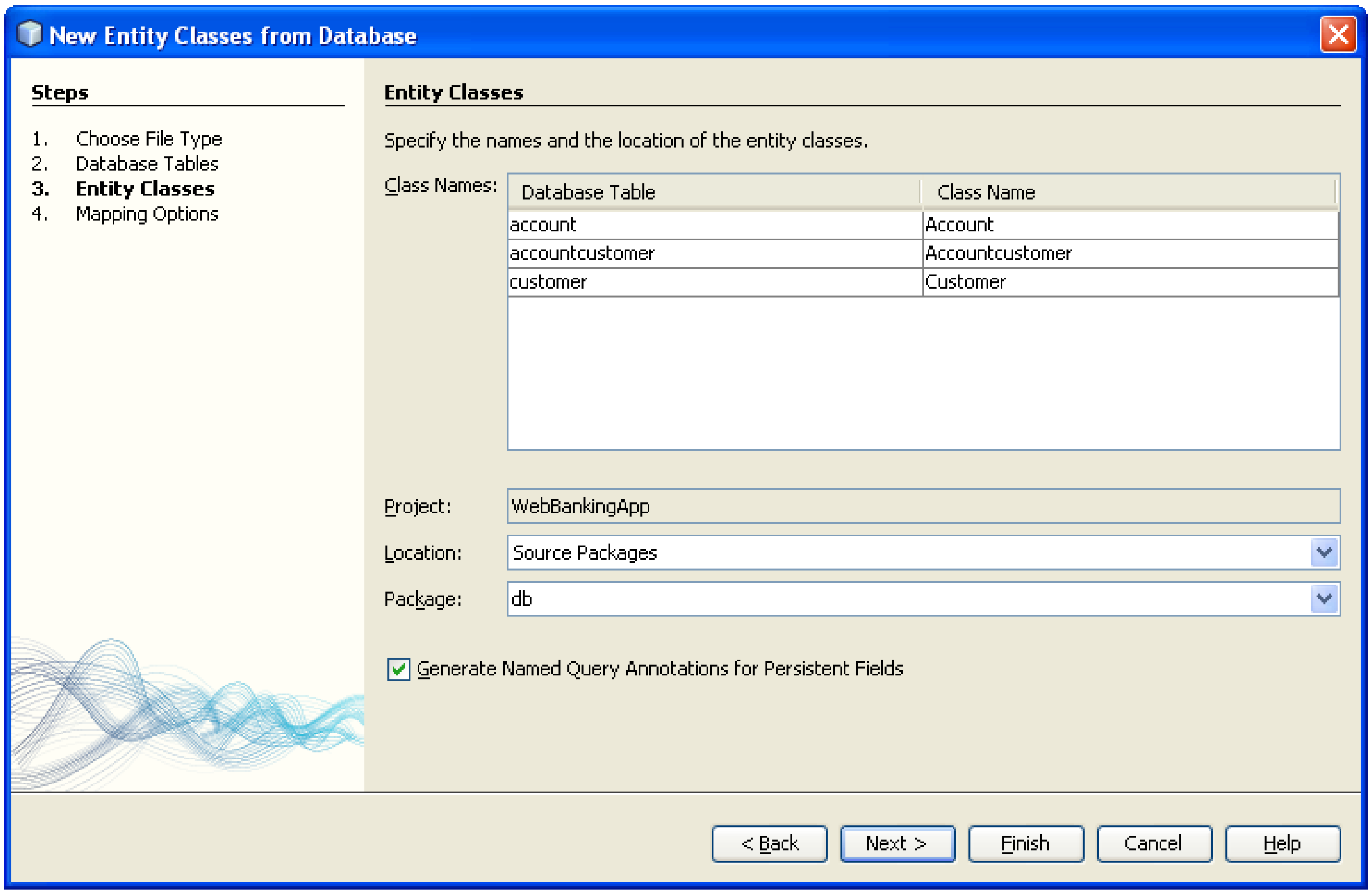
Specify the persistence provider and database for entity classes.

Persistence Provider: EclipseLink(JPA 2.0)(default)

Data Source: jndi/bank

[✓] Use Java Transaction APIs

Table Generation Strategy: ○ Create ○ Drop and Create [✓ None]



Entity Classes

Specify the names and the location of the entity classes.

Class Names:

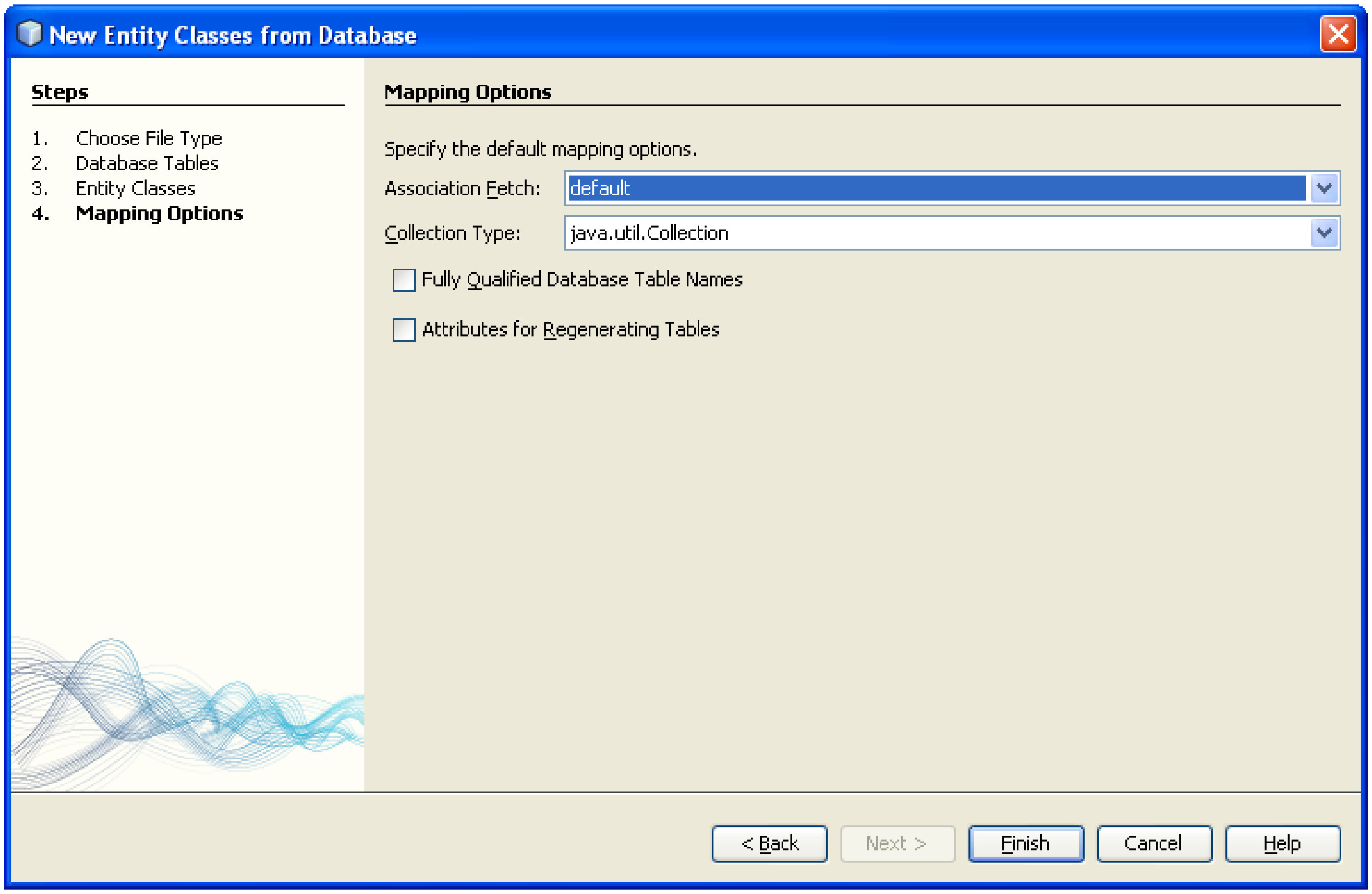
Database Table	Class Name
account	Account
accountcustomer	Accountcustomer
customer	Customer

Project: WebBankingApp

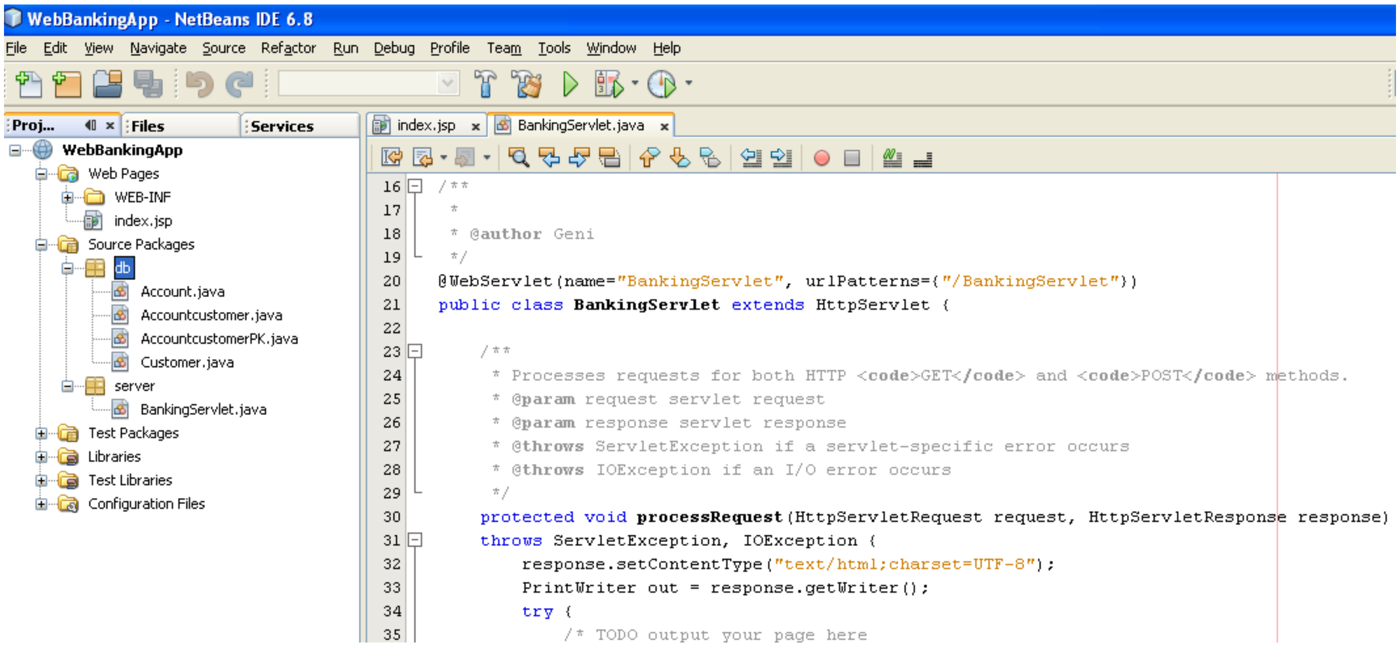
Location: Source Packages

Package: db

[✓] Generate Named Query Annotations for Persistent Fields

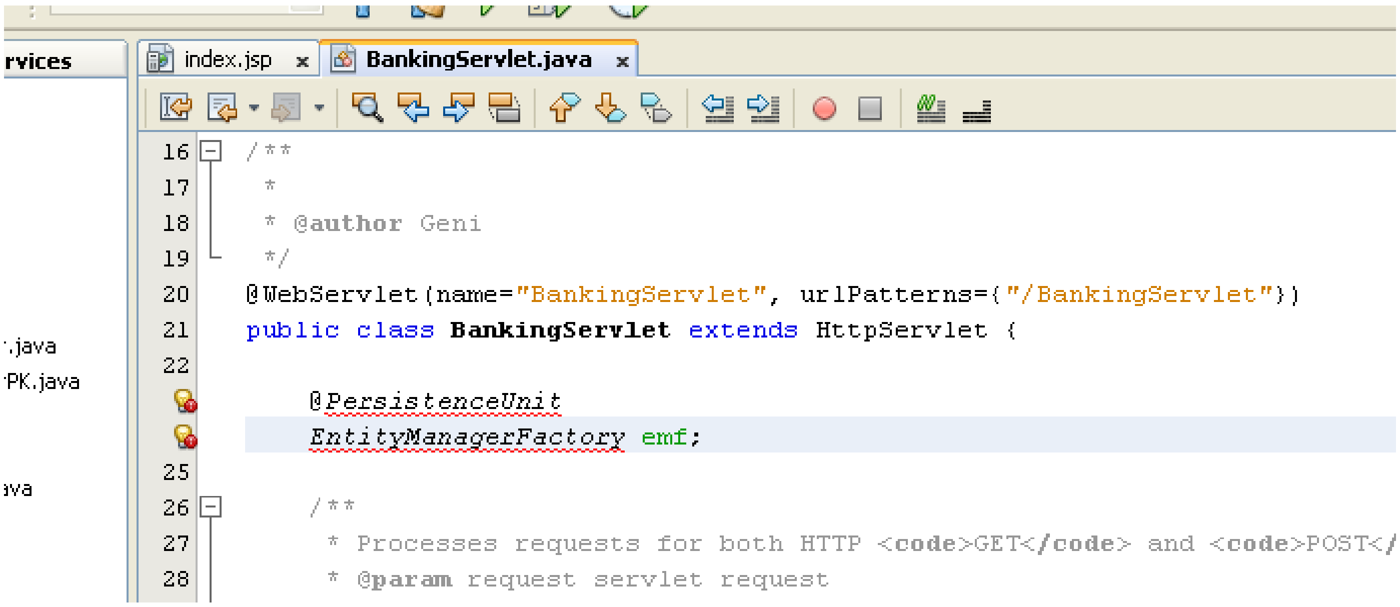

New Entity Classes from Database
Steps
1. Choose File Type
2. Database Tables
3. Entity Classes
4. Mapping Options
Mapping Options
Specify the default mapping options.
Association Fetch: default
Collection Type: java.util.Collection
Fully Qualified Database Table Names
Attributes for Regenerating Tables
< Back
Next >
Finish
Cancel
Help

The following will be generated:



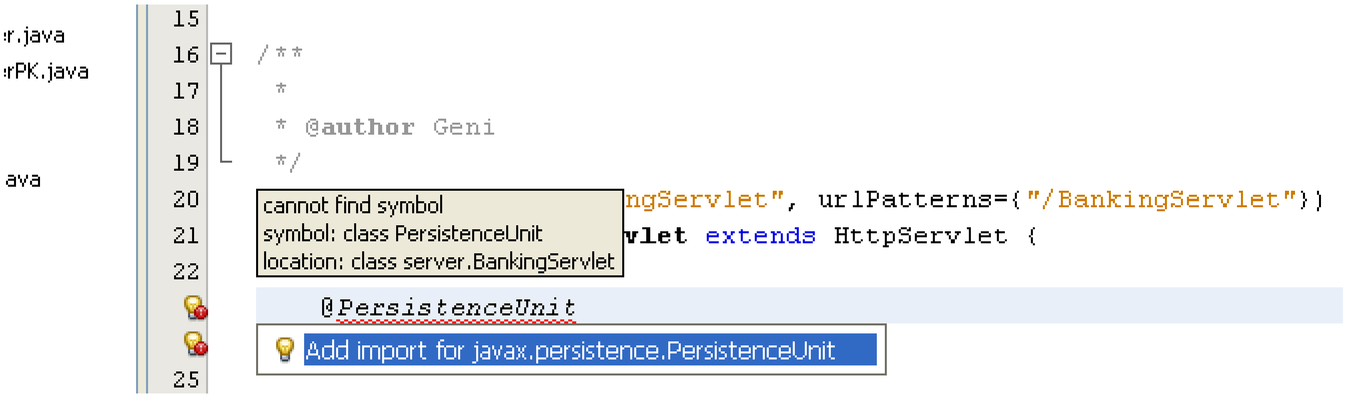
```java
/**
 *
 * @author Geni
 */
@WebServlet(name="BankingServlet", urlPatterns={"/BankingServlet"})
public class BankingServlet extends HttpServlet {

    /**
     * Processes requests for both HTTP <code>GET</code> and <code>POST</code> methods.
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    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
```

In BankingServlet code the following:



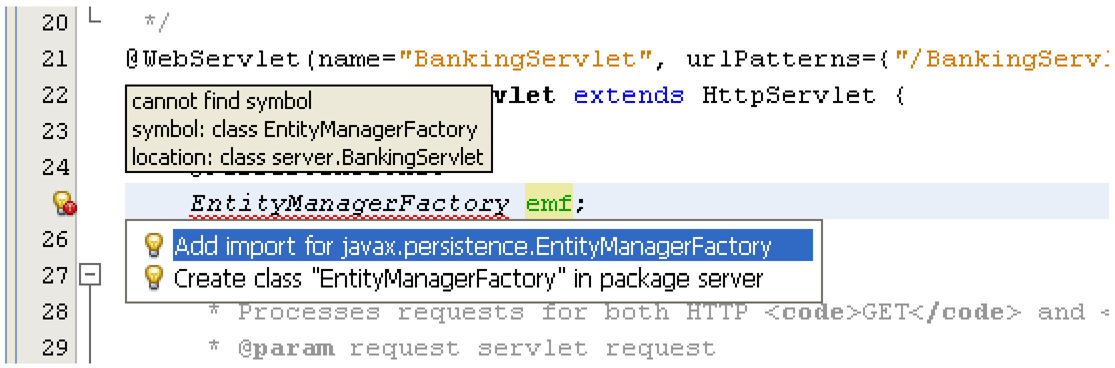
```
/**
 *
 * @author Geni
 */
@WebServlet(name="BankingServlet", urlPatterns={"/BankingServlet"})
public class BankingServlet extends HttpServlet {

    @PersistenceUnit
    EntityManagerFactory emf;

    /**
     * Processes requests for both HTTP <code>GET</code> and <code>POST</
     * @param request servlet request
```

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



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

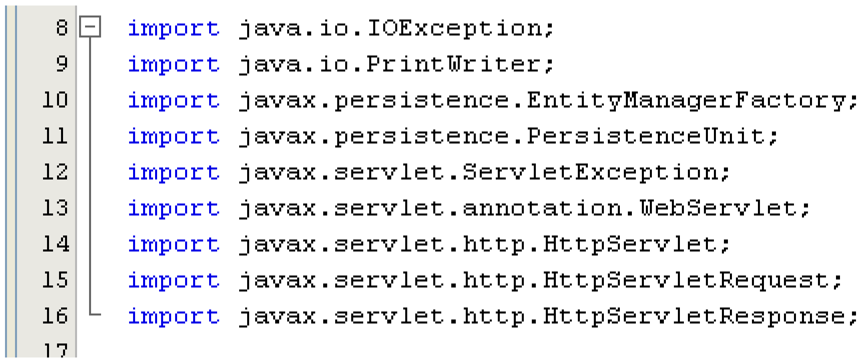
```
Add import for javax.persistence.PersistenceUnit
```



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

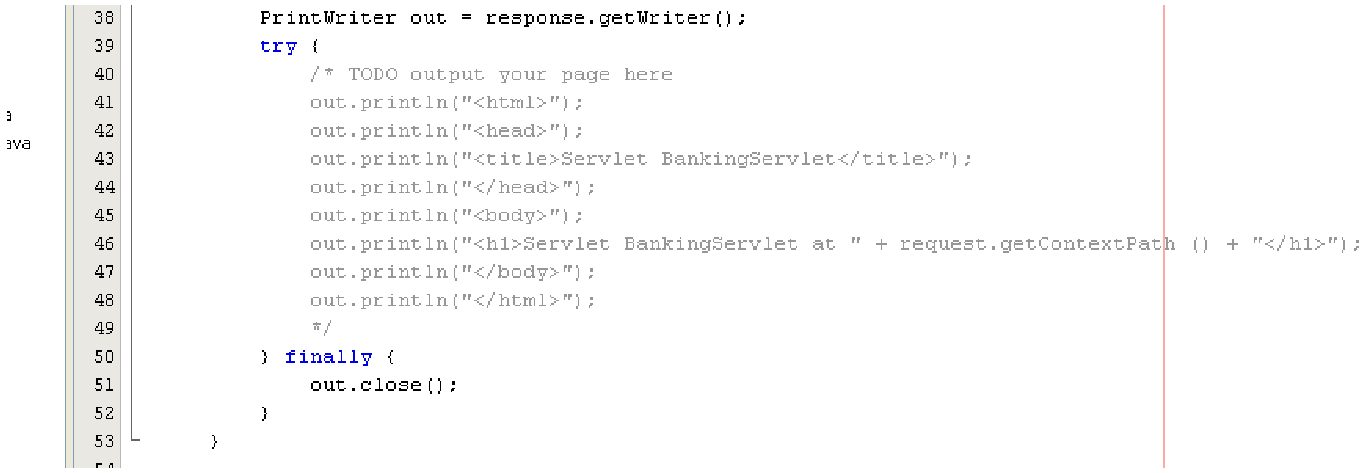
```
Add import for javax.persistence.EntityManagerFactory
Create class "EntityManagerFactory" in package server
```

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



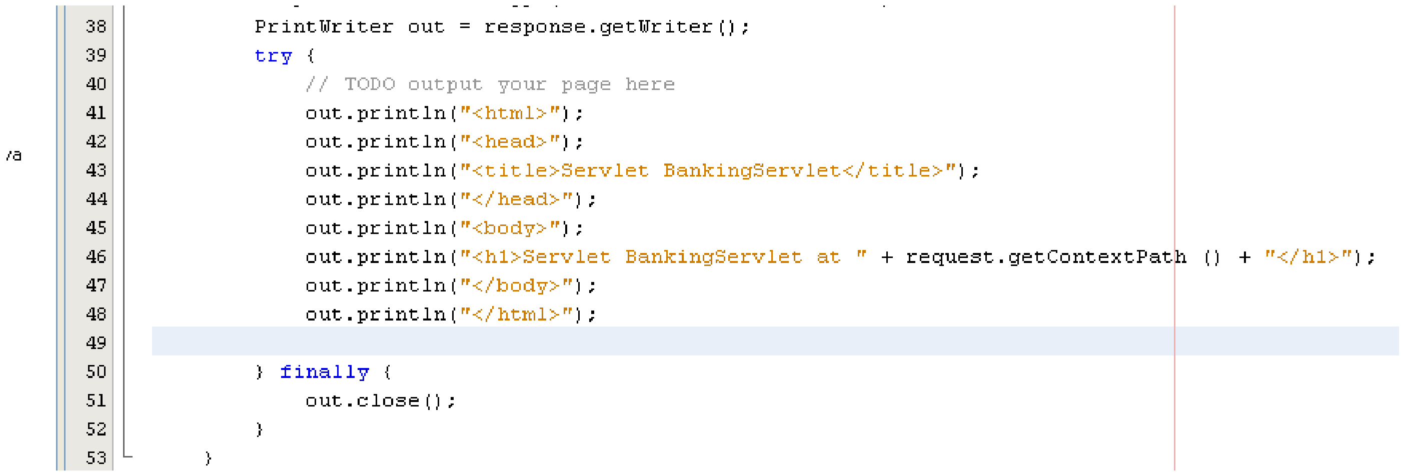
```
import java.io.IOException;
import java.io.PrintWriter;
import javax.persistence.EntityManagerFactory;
import javax.persistence.PersistenceUnit;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

Now we go the point of generating the webpage.



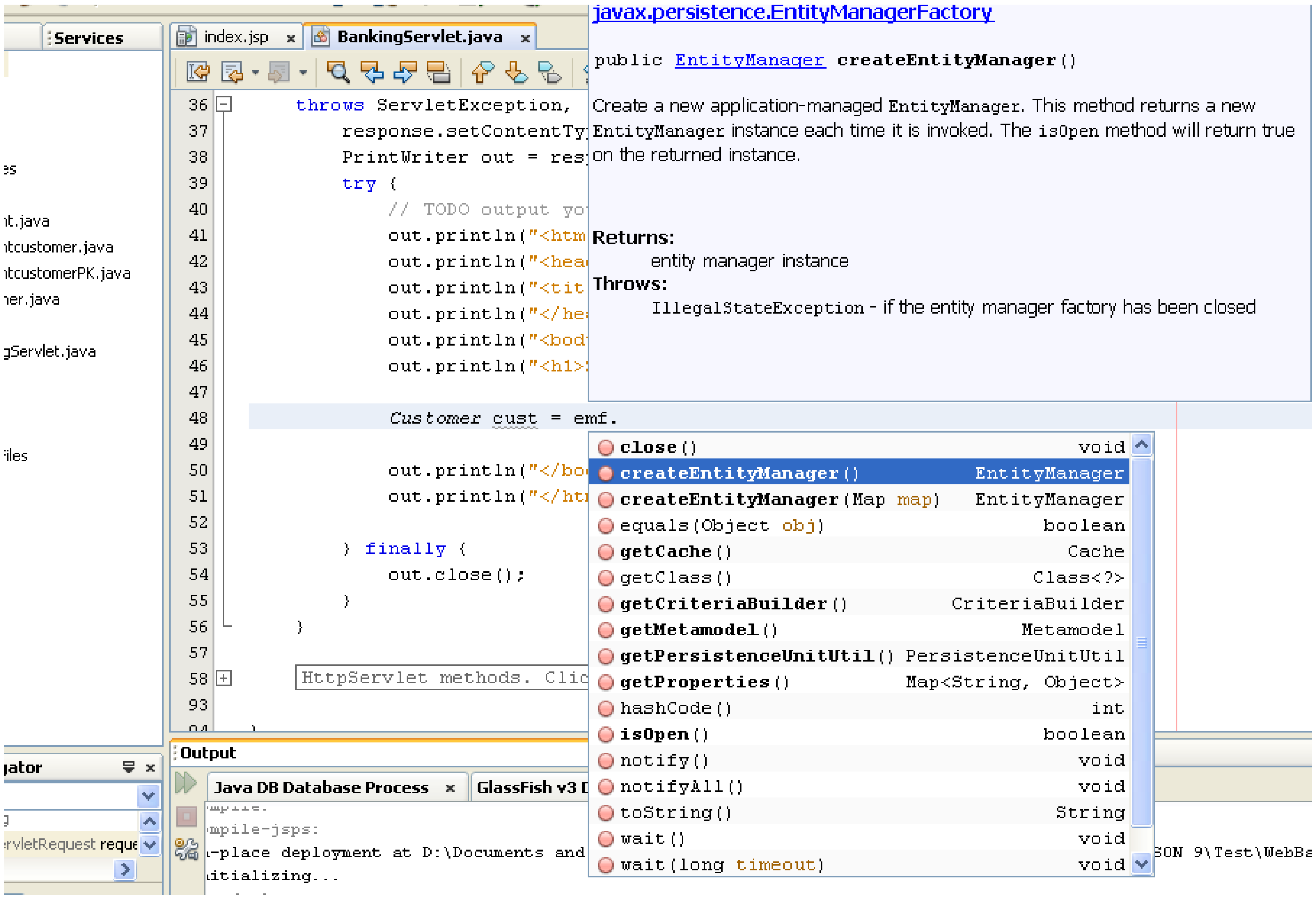
```
        PrintWriter out = response.getWriter();
        try {
            /* TODO output your page here
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet BankingServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet BankingServlet at " + request.getContextPath () + "</h1>");
            out.println("</body>");
            out.println("</html>");
            */
        } finally {
            out.close();
        }
    }
```

Uncomment the commented part as follows:



```
        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 " + request.getContextPath () + "</h1>");
            out.println("</body>");
            out.println("</html>");

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

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



```
        throws ServletException,
            response.setContentTy
            PrintWriter out = res
            try {
                // TODO output yo
                out.println("<htm
                out.println("<hea
                out.println("<tit
                out.println("</he
                out.println("<bod
                out.println("<h1>

                Customer cust = emf.

                out.println("</bo
                out.println("</ht

            } finally {
                out.close();
            }
        }

    HttpServlet methods. Clic
```

javax.persistence.EntityManagerFactory

`public EntityManager createEntityManager()`

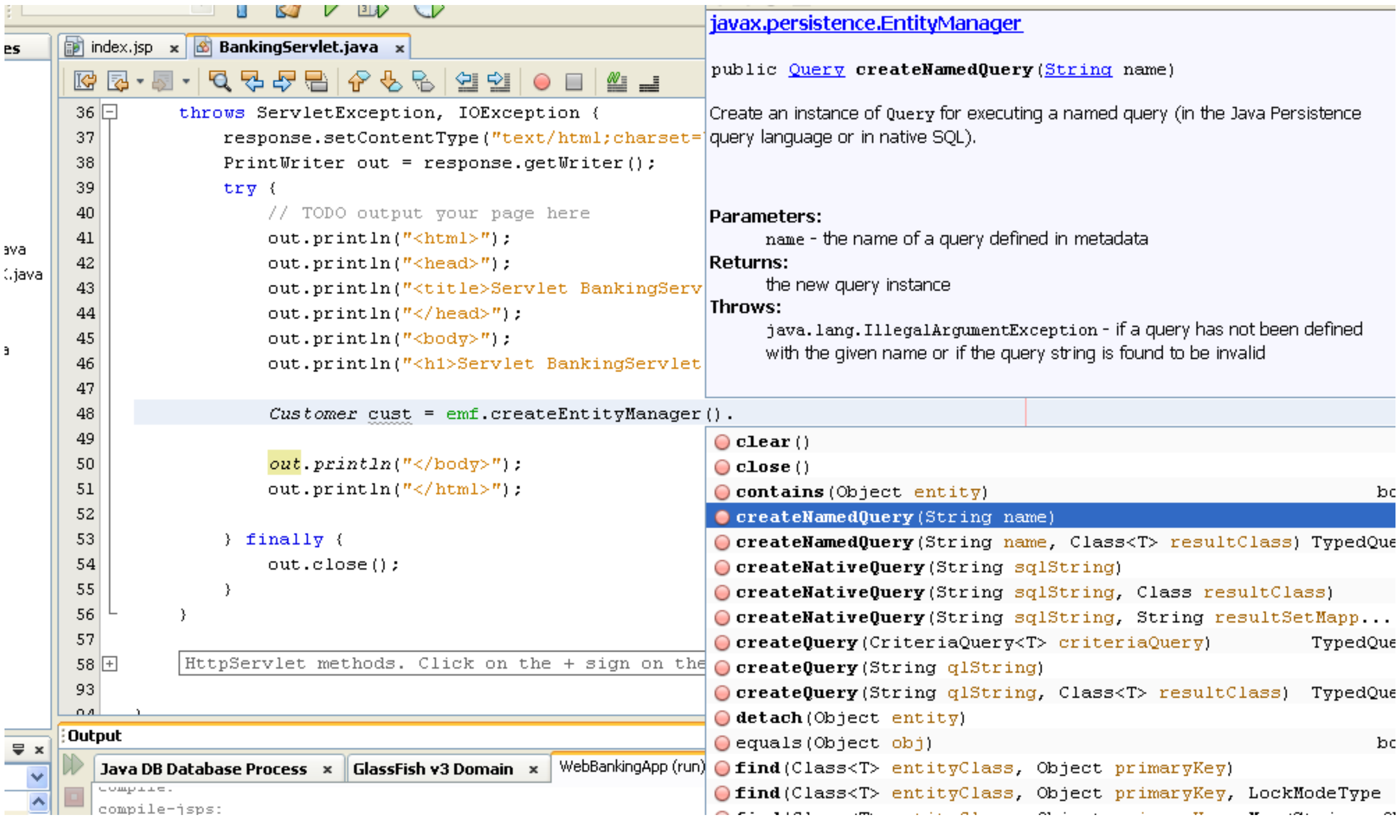
Create a new application-managed `EntityManager`. This method returns a new `EntityManager` instance each time it is invoked. The `isOpen` method will return true on the returned instance.

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

```
close()                                  void
createEntityManager()                    EntityManager
createEntityManager(Map map)             EntityManager
equals(Object obj)                       boolean
getCache()                               Cache
getClass()                               Class<?>
getCriteriaBuilder()                     CriteriaBuilder
getMetamodel()                           Metamodel
getPersistenceUnitUtil() PersistenceUnitUtil
getProperties()             Map<String, Object>
hashCode()                               int
isOpen()                                 boolean
notify()                                 void
notifyAll()                              void
toString()                               String
wait()                                   void
wait(long timeout)                       void
```

```
mpile-jsps:
-place deployment at D:\Documents and            SON 9\Test\WebBa
itializing...
```

Then:



```
        throws ServletException, IOException {
            response.setContentType("text/html;charset=
            PrintWriter out = response.getWriter();
            try {
                // TODO output your page here
                out.println("<html>");
                out.println("<head>");
                out.println("<title>Servlet BankingServ
                out.println("</head>");
                out.println("<body>");
                out.println("<h1>Servlet BankingServlet

                Customer cust = emf.createEntityManager().

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

            } finally {
                out.close();
            }
        }

        HttpServlet methods. Click on the + sign on the
```

javax.persistence.EntityManager

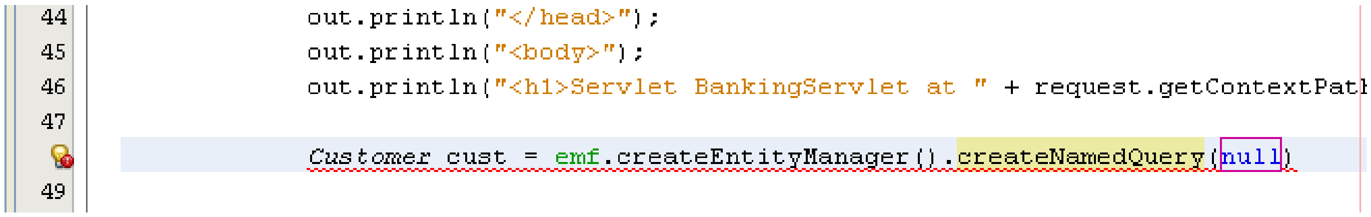
```
public Query createNamedQuery(String name)
```

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

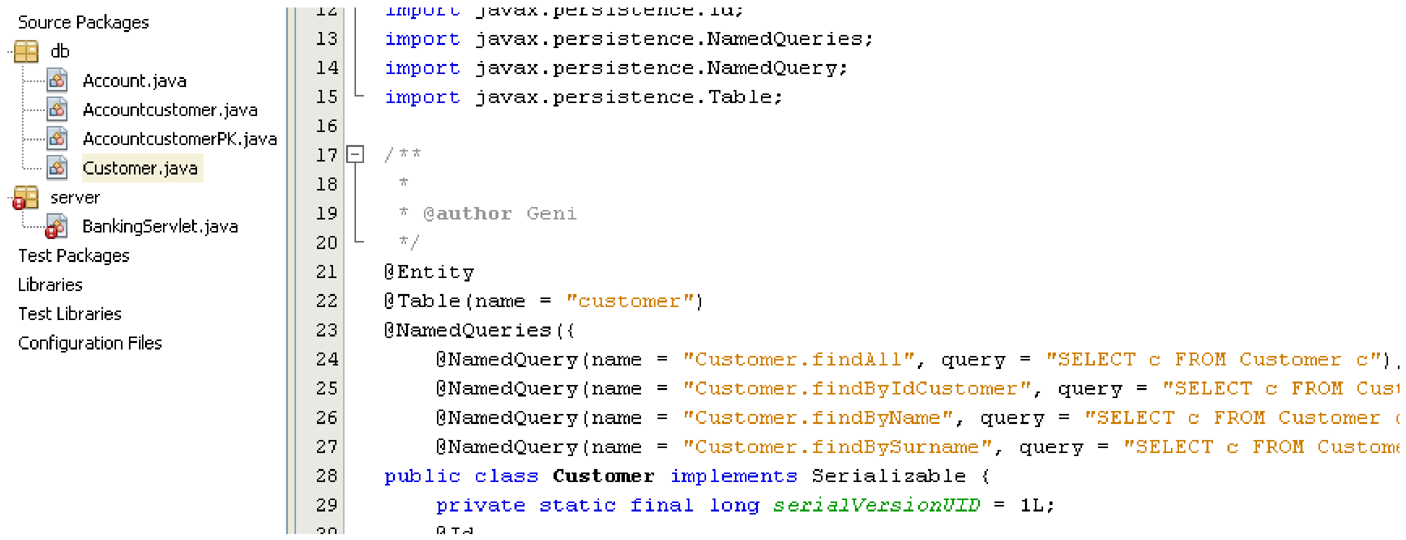
```
clear()
close()
contains(Object entity)                                   bo
createNamedQuery(String name)
createNamedQuery(String name, Class<T> resultClass) TypedQue
createNativeQuery(String sqlString)
createNativeQuery(String sqlString, Class resultClass)
createNativeQuery(String sqlString, String resultSetMapp...
createQuery(CriteriaQuery<T> criteriaQuery)         TypedQue
createQuery(String qlString)
createQuery(String qlString, Class<T> resultClass)  TypedQue
detach(Object entity)
equals(Object obj)                                        bo
find(Class<T> entityClass, Object primaryKey)
find(Class<T> entityClass, Object primaryKey, LockModeType
```

You have now:



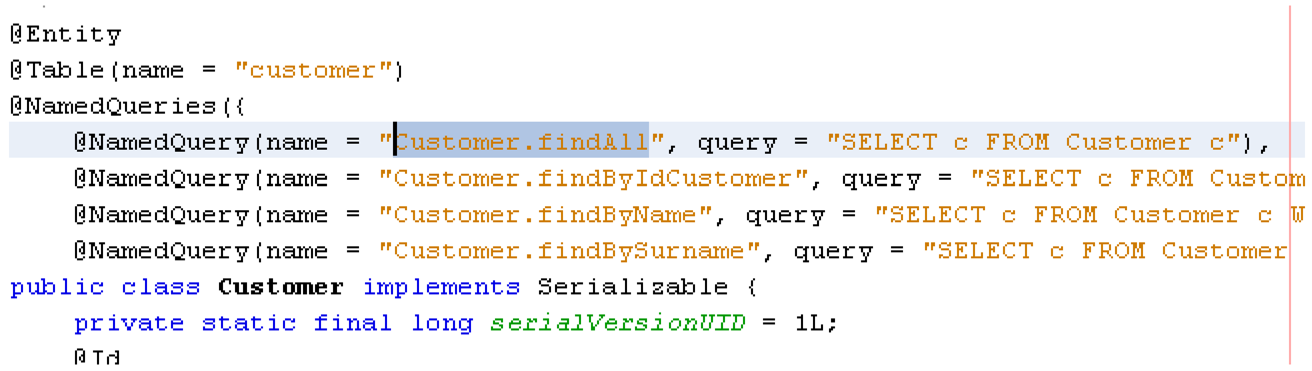
```
                out.println("</head>");
                out.println("<body>");
                out.println("<h1>Servlet BankingServlet at " + request.getContextPat

                Customer cust = emf.createEntityManager().createNamedQuery(null)
```

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

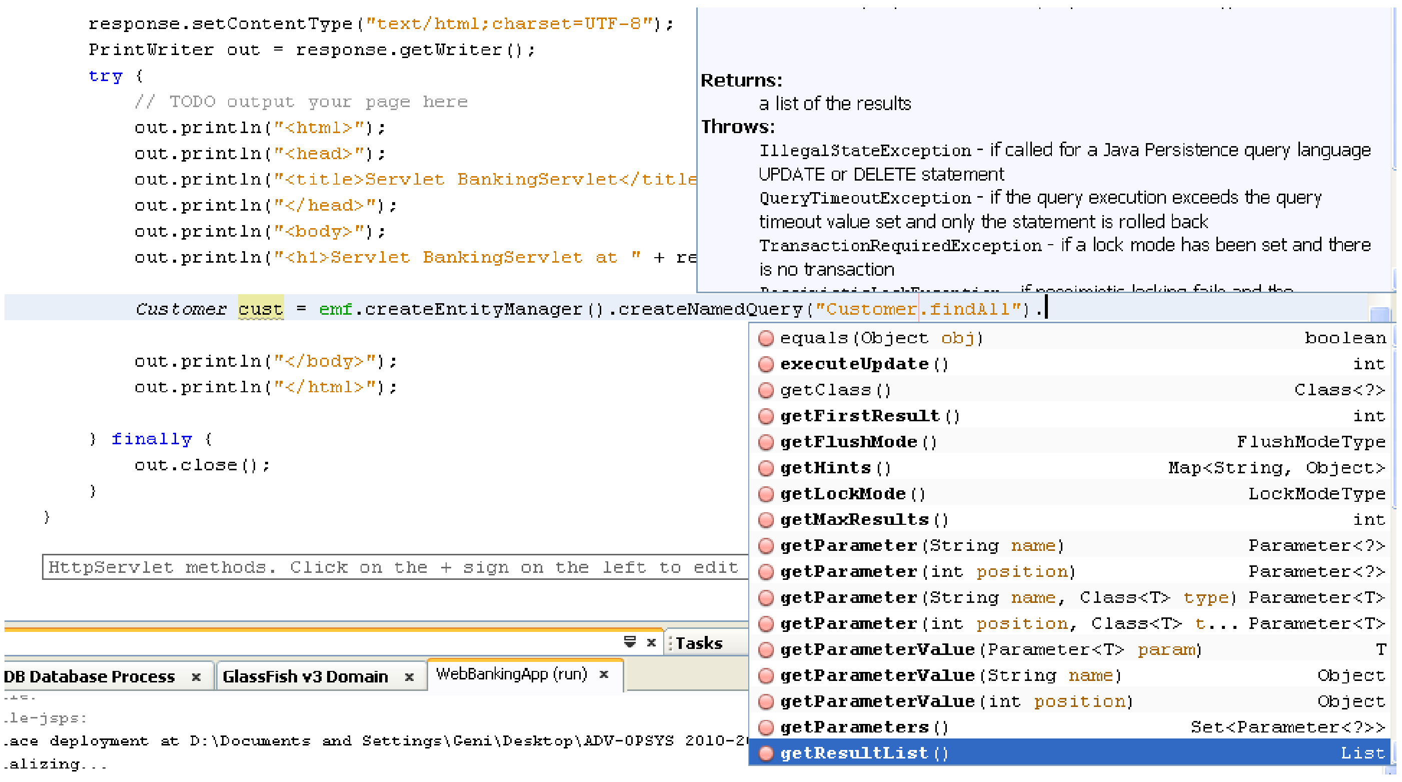

```
Source Packages
  db
    Account.java
    Accountcustomer.java
    AccountcustomerPK.java
    Customer.java
  server
    BankingServlet.java
Test Packages
Libraries
Test Libraries
Configuration Files
```

```
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

/**
 *
 * @author Geni
 */
@Entity
@Table(name = "customer")
@NamedQueries({
    @NamedQuery(name = "Customer.findAll", query = "SELECT c FROM Customer c"),
    @NamedQuery(name = "Customer.findByIdCustomer", query = "SELECT c FROM Cust
    @NamedQuery(name = "Customer.findByName", query = "SELECT c FROM Customer c
    @NamedQuery(name = "Customer.findBySurname", query = "SELECT c FROM Custome
public class Customer implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
```

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;
    @Id
```

Paste what you copied from Customer, in the following window in the servlet. Now get the results from th 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

            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
```

Returns:
a list of the results

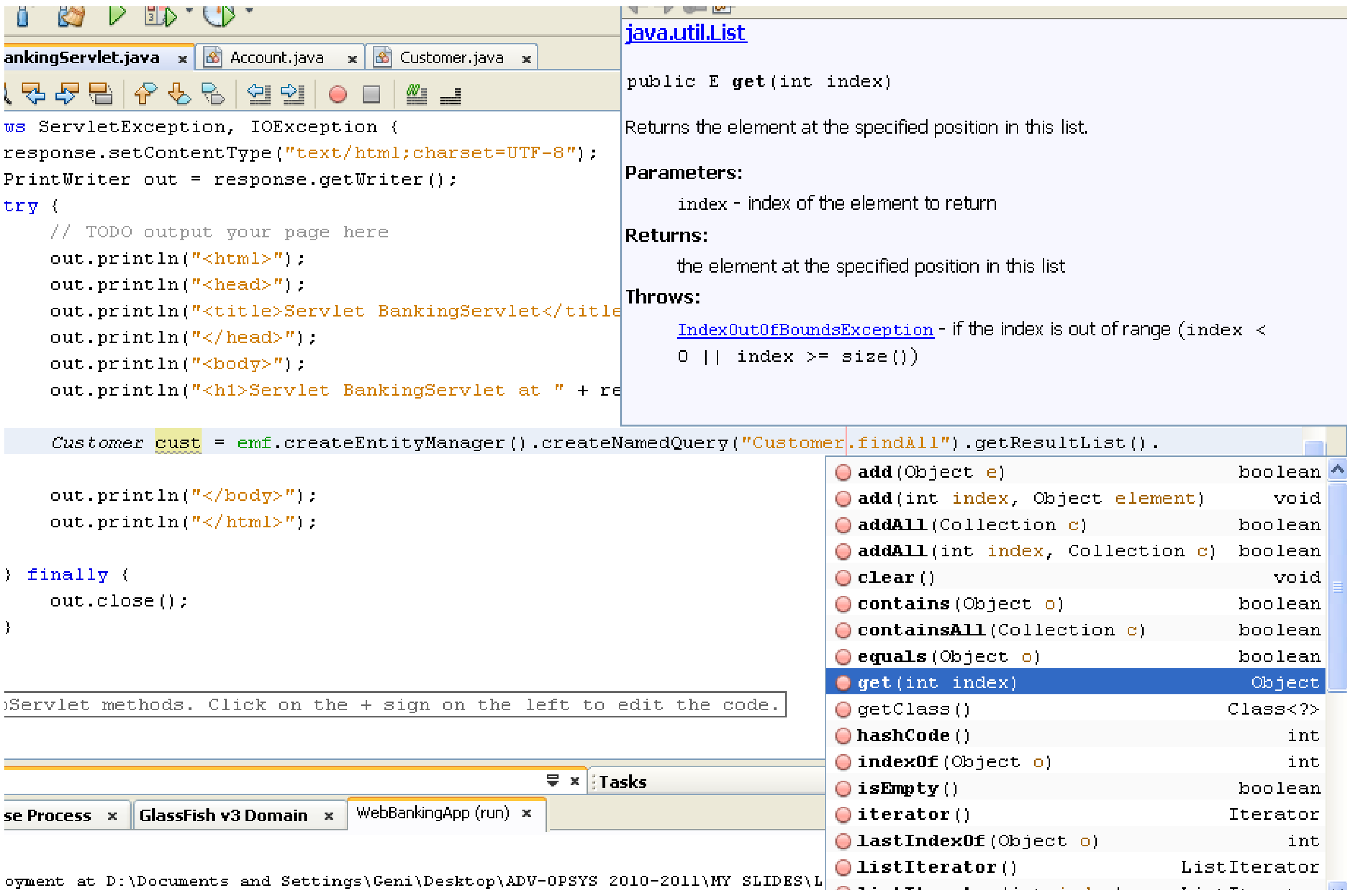
Throws:
`IllegalStateException` - if called for a Java Persistence query language UPDATE or DELETE statement
`QueryTimeoutException` - if the query execution exceeds the query timeout value set and only the statement is rolled back
`TransactionRequiredException` - if a lock mode has been set and there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

```
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 × | GlassFish v3 Domain × | WebBankingApp (run) × | Tasks

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

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


```
ws 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 BankingServlet</title
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Servlet BankingServlet at " + re

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

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

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

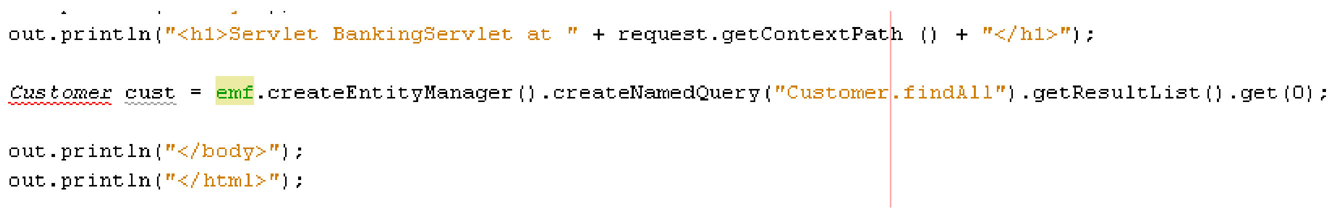
```
java.util.List

public E get(int index)

Returns the element at the specified position in this list.

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())
```

```
add(Object e)                          boolean
add(int index, Object element)            void
addAll(Collection c)                   boolean
addAll(int index, Collection c)        boolean
clear()                                   void
contains(Object o)                     boolean
containsAll(Collection c)              boolean
equals(Object o)                       boolean
get(int index)                          Object
getClass()                            Class<?>
hashCode()                                 int
indexOf(Object o)                          int
isEmpty()                              boolean
iterator()                            Iterator
lastIndexOf(Object o)                      int
listIterator()                    ListIterator
```

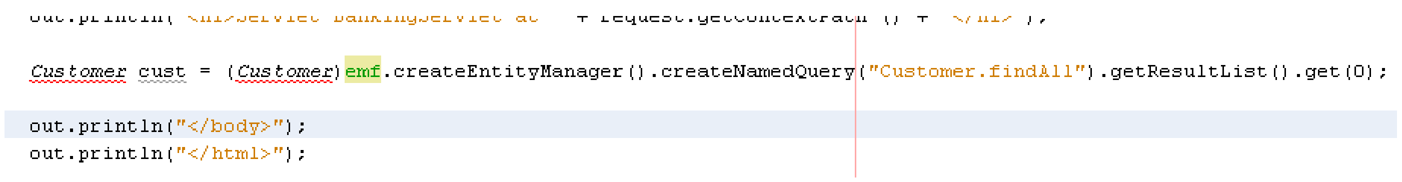
```
)Servlet methods. Click on the + sign on the left to edit the code.
```

```
oyment at D:\Documents and Settings\Geni\Desktop\ADV-OPSYS 2010-2011\MY SLIDES\L
```

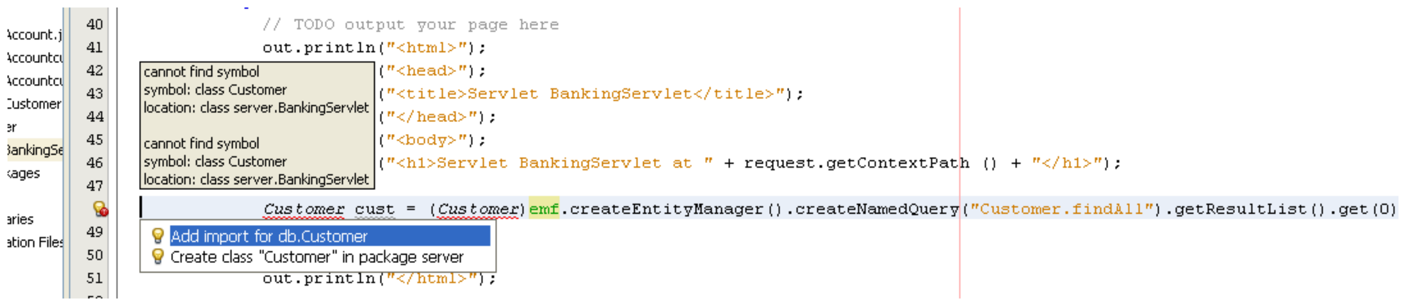
Now the code is:


```
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 aadding (Customer) before the statement::


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

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

Now arrange the imports as follows:

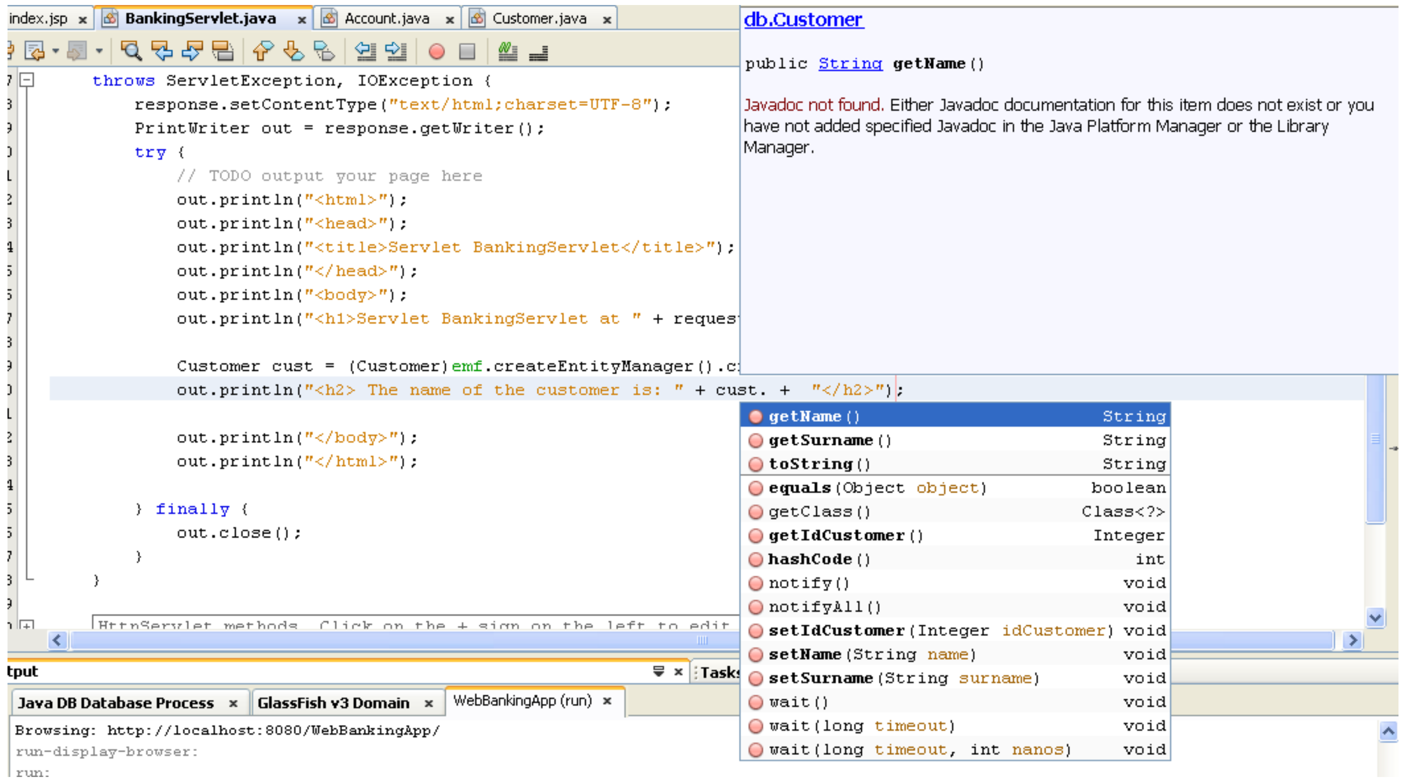


```
            // TODO output your page here
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet BankingServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet BankingServlet at " + request.getContextPath () + "</h1>");

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

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

```
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:



```
    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 BankingServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet BankingServlet at " + reques

            Customer cust = (Customer)emf.createEntityManager().c
            out.println("<h2> The name of the customer is: " + cust. + "</h2>");

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

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

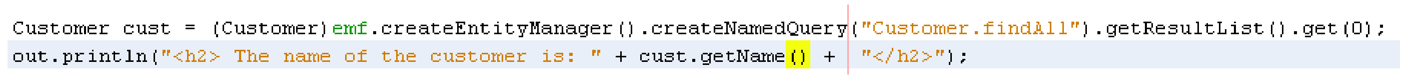
```
db.Customer

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
```

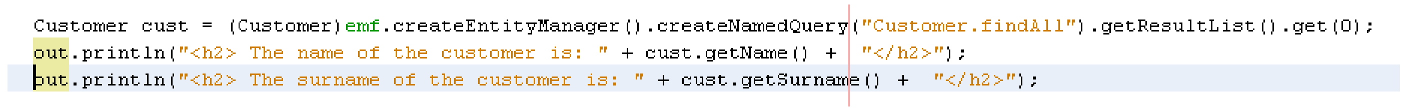
```
Browsing: http://localhost:8080/WebBankingApp/
run-display-browser:
run:
```

Now the code looks as follows:



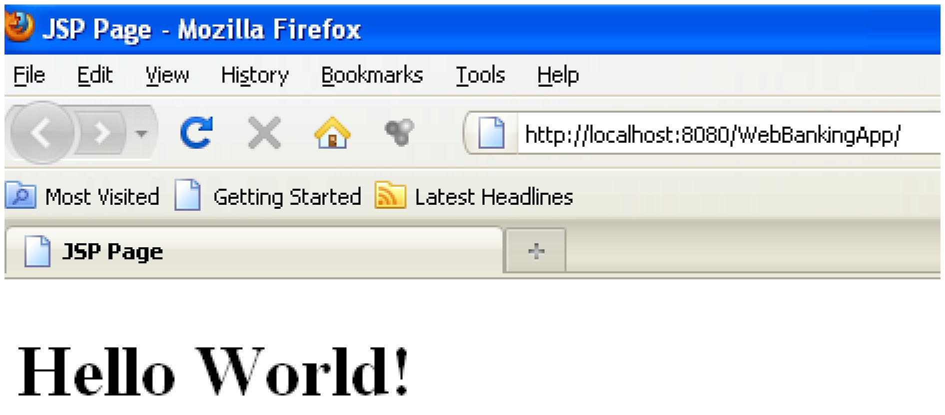
```
Customer cust = (Customer)emf.createEntityManager().createNamedQuery("Customer.findAll").getResultList().get(0);
out.println("<h2> The name of the customer is: " + cust.getName() + "</h2>");
```

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.

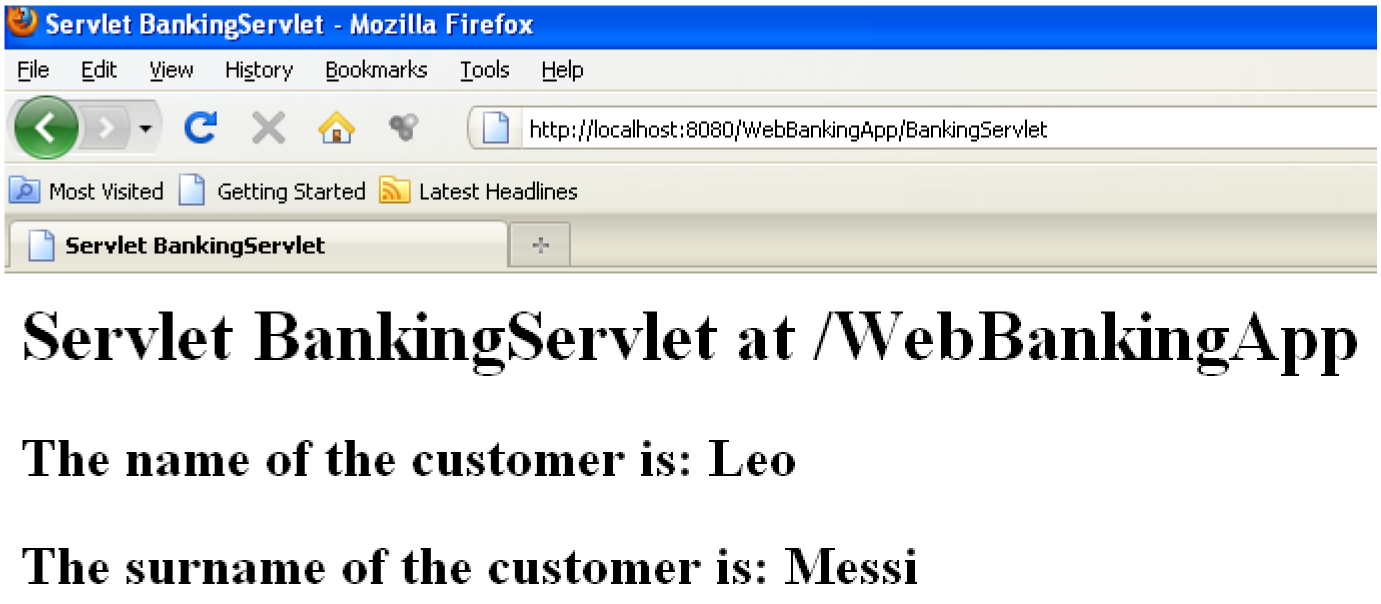

JSP Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebBankingApp/
Most Visited Getting Started Latest Headlines
JSP Page

Hello World!

Add the servlet name in the browser:

http://localhost:8080/WebBankingApp/BankingServlet

The following will appear:


Servlet BankingServlet - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebBankingApp/BankingServlet
Most Visited Getting Started Latest Headlines
Servlet BankingServlet

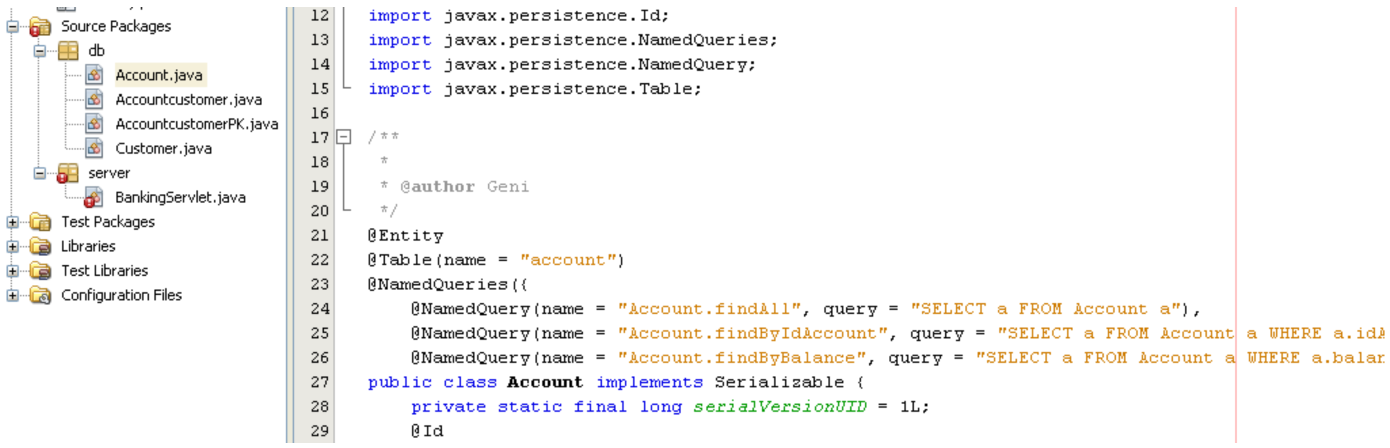
Servlet BankingServlet at /WebBankingApp

The name of the customer is: Leo

The surname of the customer is: Messi

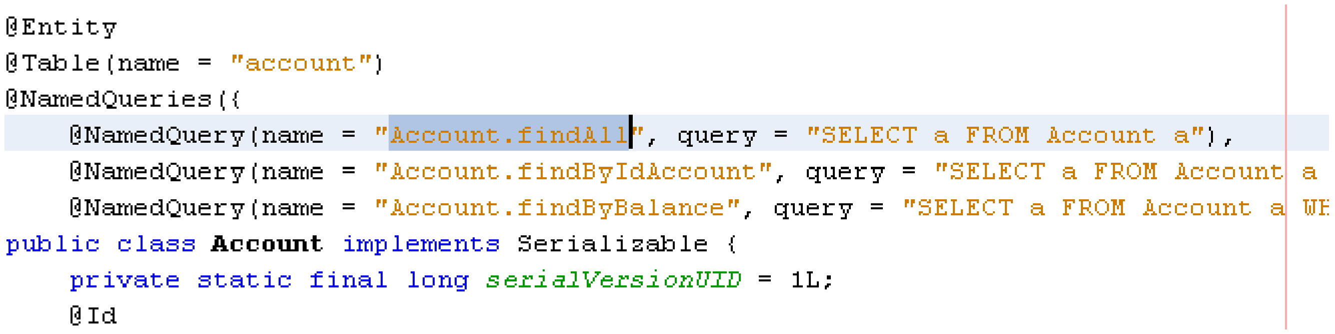
If we want to take some data about the accounts:

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



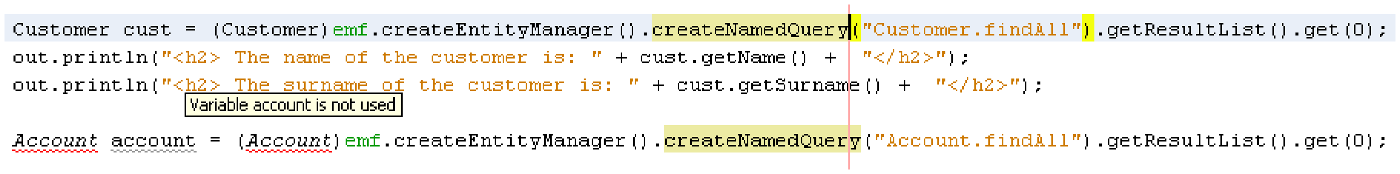
```
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

/**
 *
 * @author Geni
 */
@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.idA
    @NamedQuery(name = "Account.findByBalance", query = "SELECT a FROM Account a WHERE a.balan
public class Account implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
```

Copy “Account



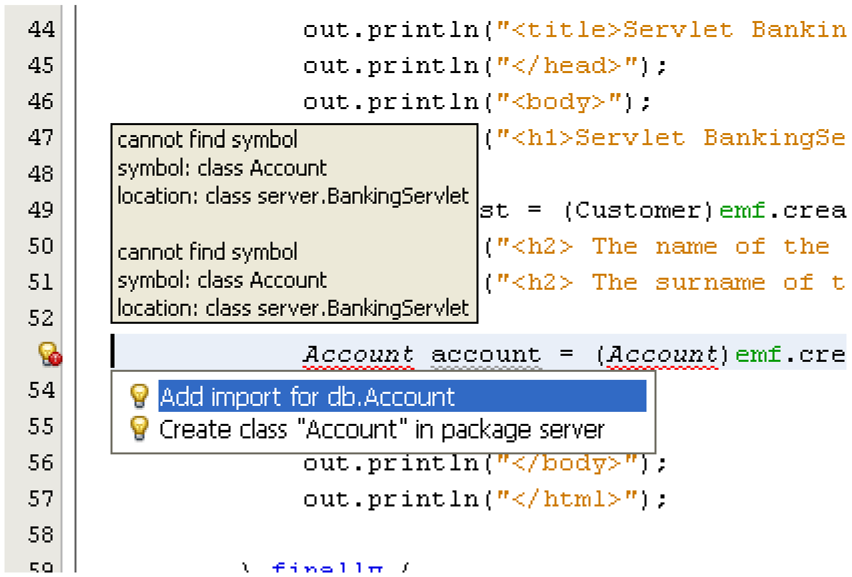
```
@Entity
@Table(name = "account")
@NamedQueries({
    @NamedQuery(name = "Account.findAll", query = "SELECT a FROM Account a"),
    @NamedQuery(name = "Account.findByIdAccount", query = "SELECT a FROM Account a
    @NamedQuery(name = "Account.findByBalance", query = "SELECT a FROM Account a WH
public 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:


```
                out.println("<title>Servlet Bankin
                out.println("</head>");
                out.println("<body>");
                out.println("<h1>Servlet BankingSe

                ...st = (Customer)emf.crea
                out.println("<h2> The name of the
                out.println("<h2> The surname of t

                Account account = (Account)emf.cre
```

```
cannot find symbol
symbol: class Account
location: class server.BankingServlet

cannot find symbol
symbol: class Account
location: class server.BankingServlet
```

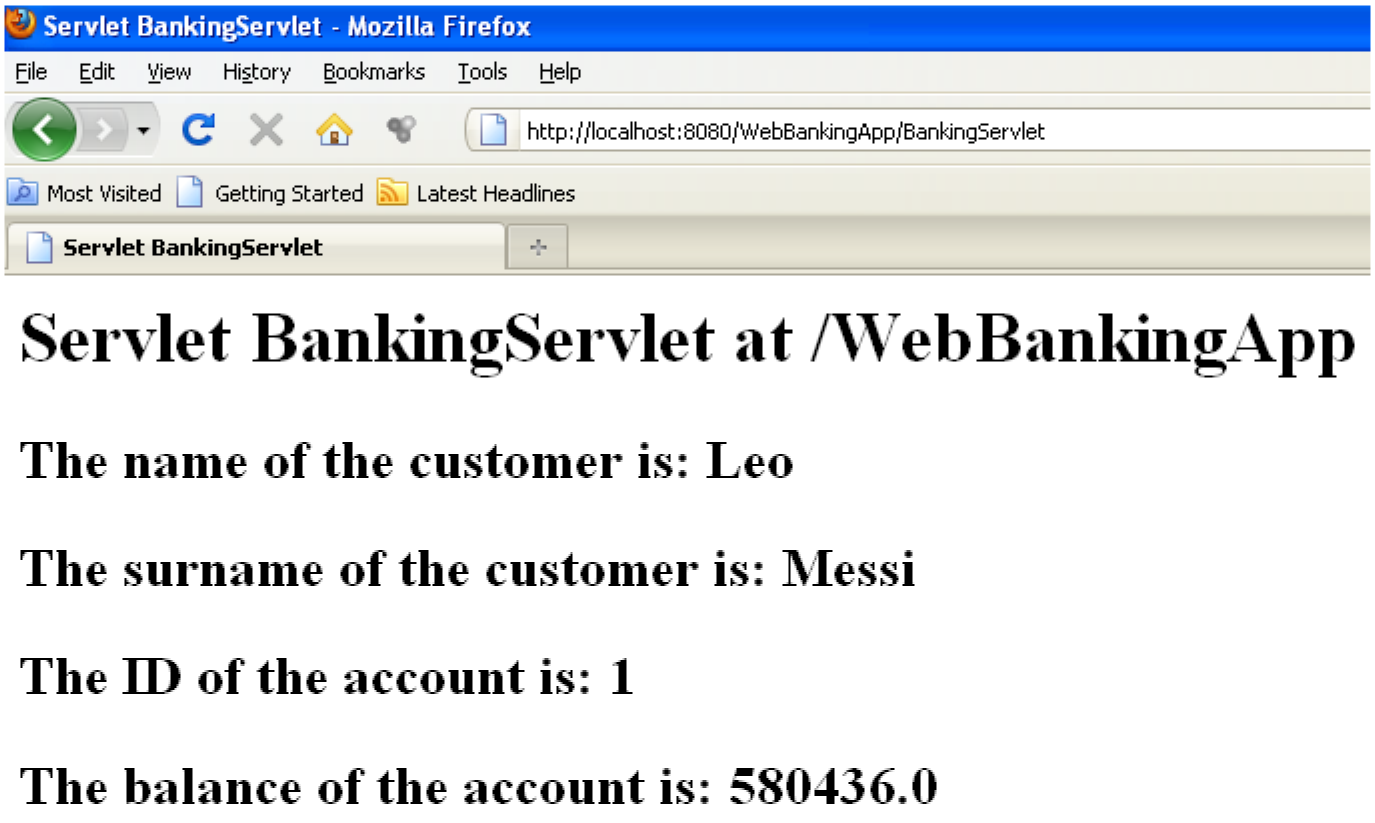
Add import for db.Account
Create class "Account" in package server

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

            } finally {
```

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

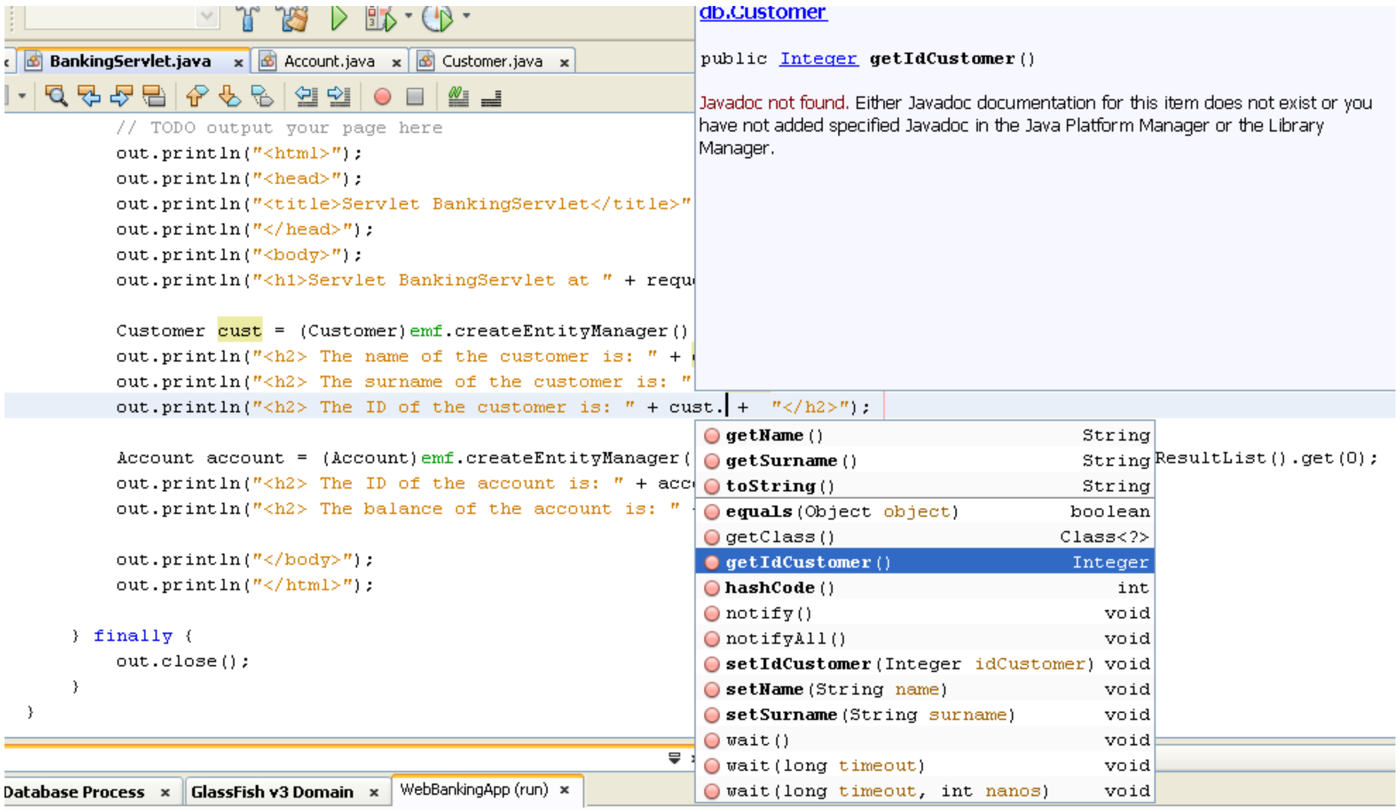
```
Account account = (Account)emf.createEntityManager().createNamedQuery("Account.findAll").getResultList().get(0);
out.println("<h2> The ID of the account is: " + account.getIdAccount() +  "</h2>");
out.println("<h2> The balance of the account is: " + account.getBalance() +  "</h2>");
```

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


Servlet BankingServlet - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebBankingApp/BankingServlet
Most Visited Getting Started Latest Headlines
Servlet BankingServlet

```
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:



```
        // 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 " + requ

        Customer cust = (Customer)emf.createEntityManager()
        out.println("<h2> The name of the customer is: " +
        out.println("<h2> The surname of the customer is: "
        out.println("<h2> The ID of the customer is: " + cust.| +  "</h2>");

        Account account = (Account)emf.createEntityManager(
        out.println("<h2> The ID of the account is: " + acc
        out.println("<h2> The balance of the account is: "

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

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

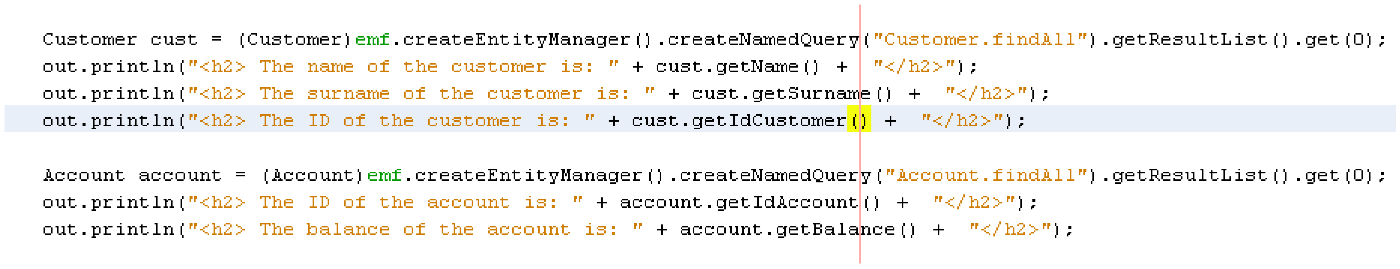
db.Customer

```
public Integer getIdCustomer()
```

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.

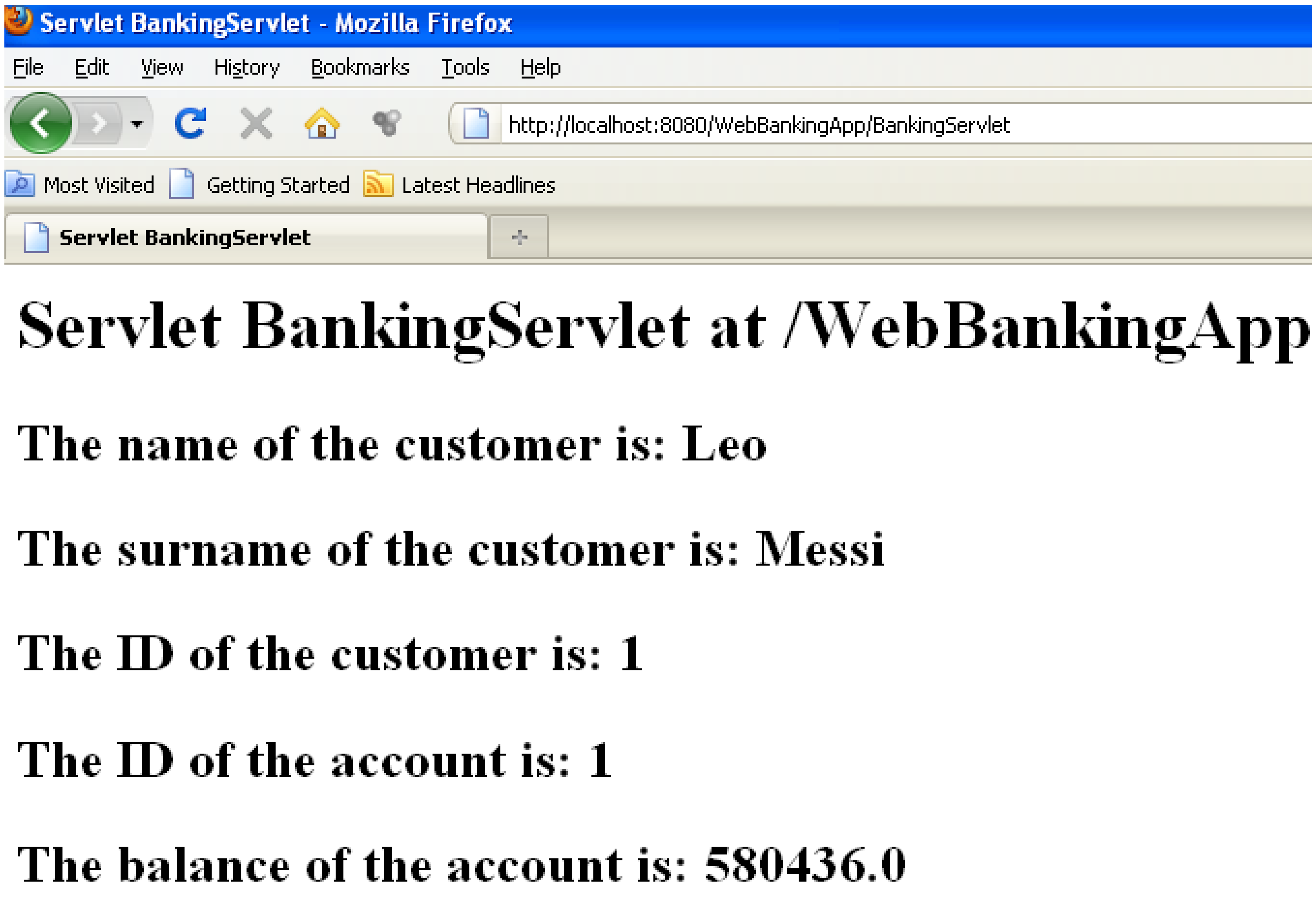
```
getName()                                 String
getSurname()                              String
toString()                                String
equals(Object object)                     boolean
getClass()                                Class<?>
getIdCustomer()                           Integer
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
```

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 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebBankingApp/BankingServlet
Most Visited Getting Started Latest Headlines
Servlet BankingServlet

Servlet BankingServlet at /WebBankingApp

The name of the customer is: Leo

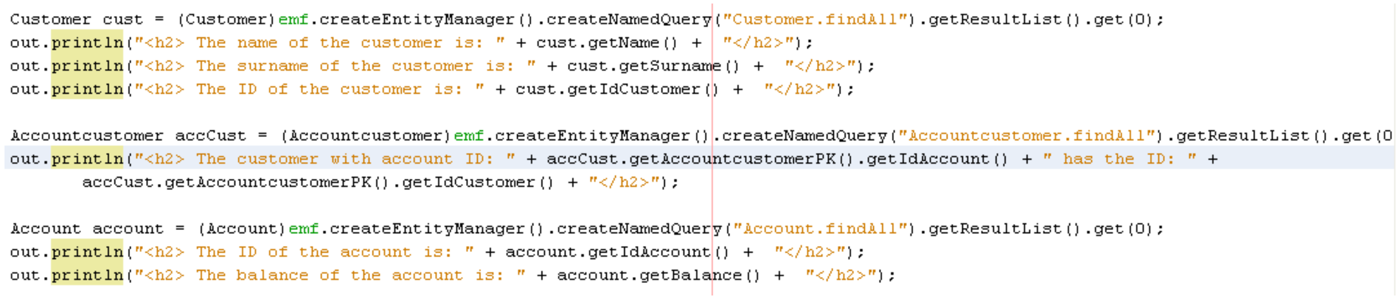
The surname of the customer is: Messi

The ID of the customer is: 1

The ID of the account is: 1

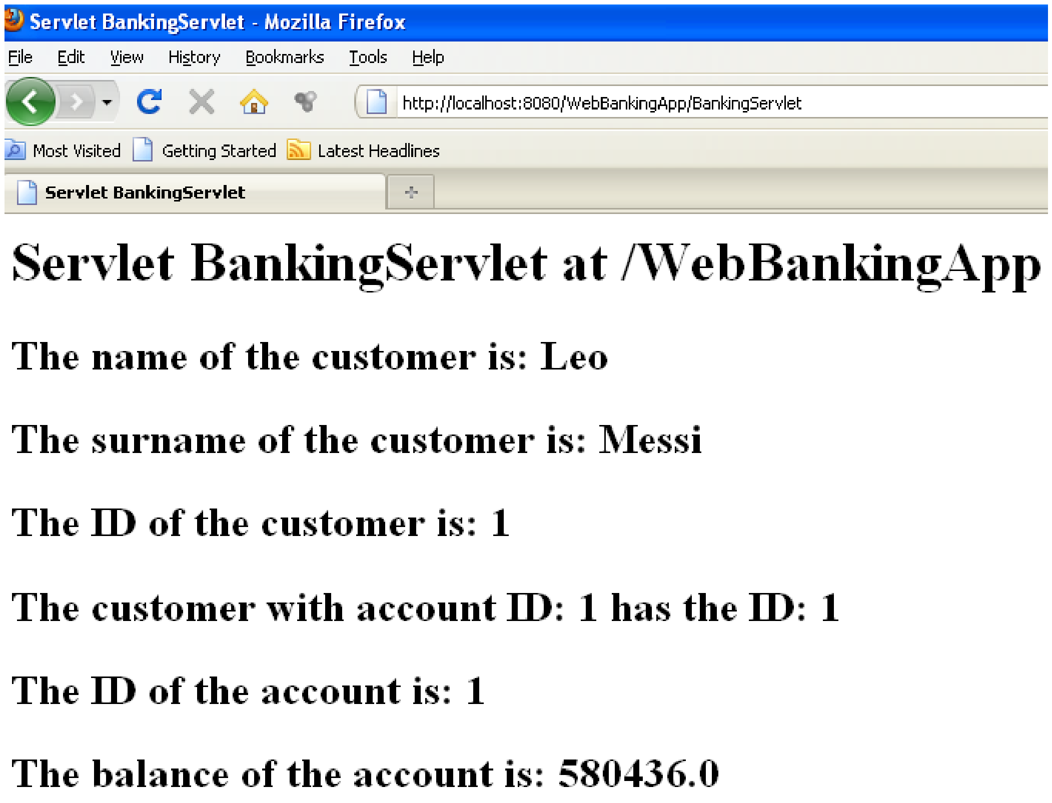
The balance of the account is: 580436.0

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 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebBankingApp/BankingServlet
Most Visited Getting Started Latest Headlines
Servlet BankingServlet

```
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
```

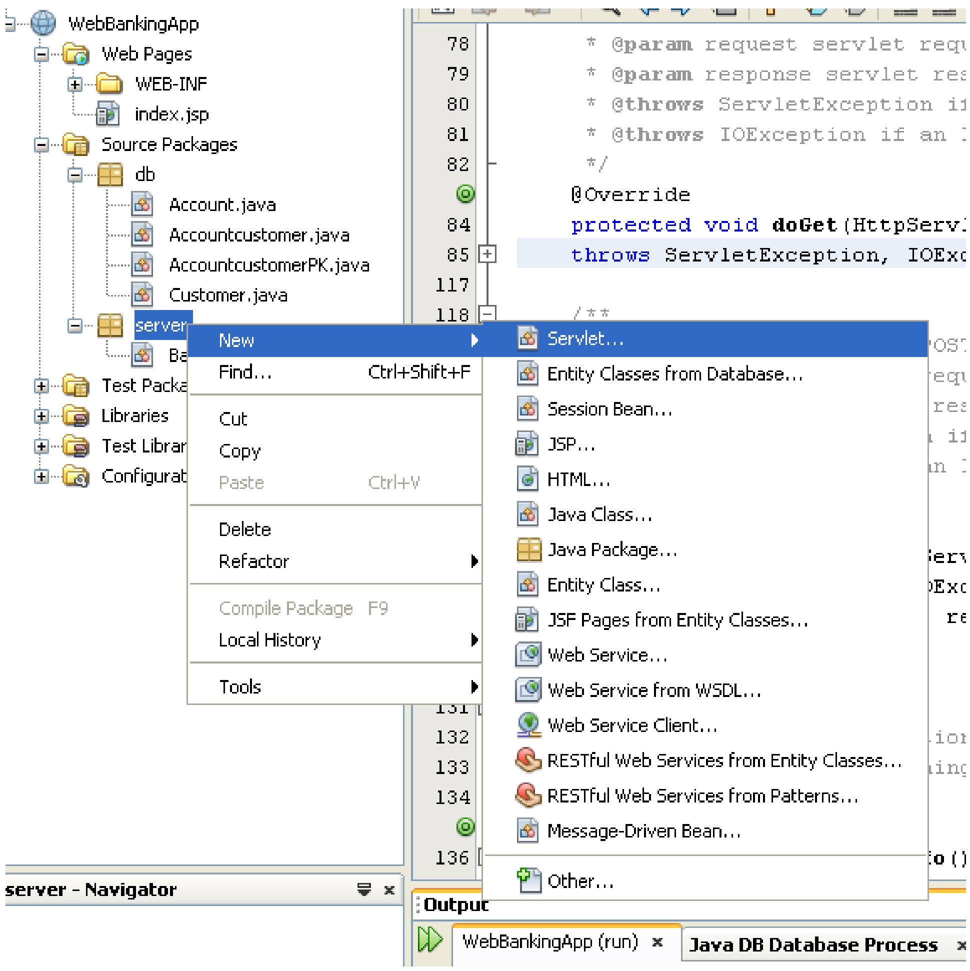
3. Developing the client interface of the web banking application

Developing a client interface to ask for the accounts of a customer with a certain ID.

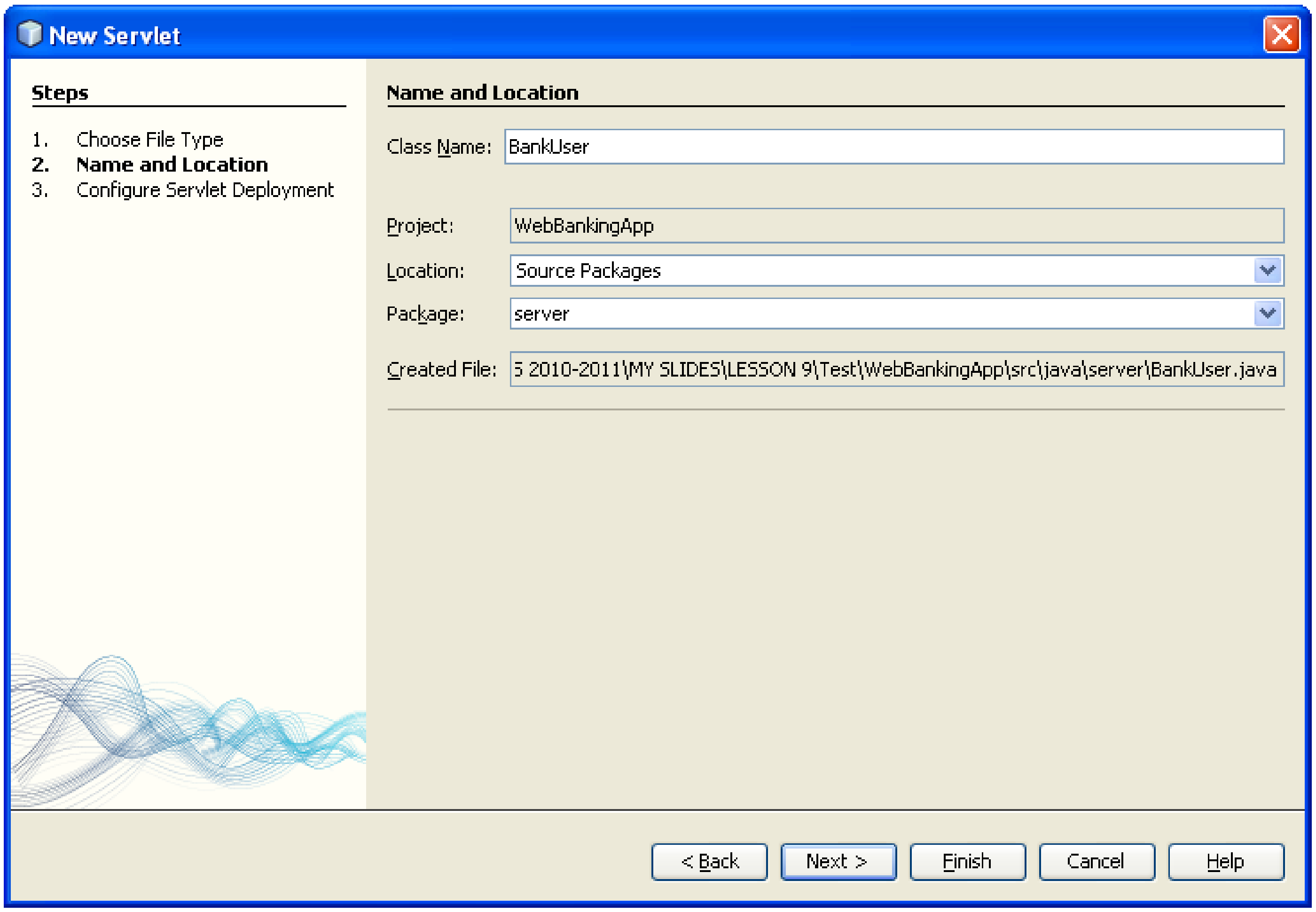
First of all we should develop a servlet that takes the input from the user and sends it to a processing servlet that reads the data from the database.

To complete this task perform the following steps:

Create a Servlet:



```
     * @param request servlet requ
     * @param response servlet res
     * @throws ServletException i
     * @throws IOException if an I
     */
    @Override
    protected void doGet(HttpServl
    throws ServletException, IOExc

    /**
```


New Servlet

Steps

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

Name and Location

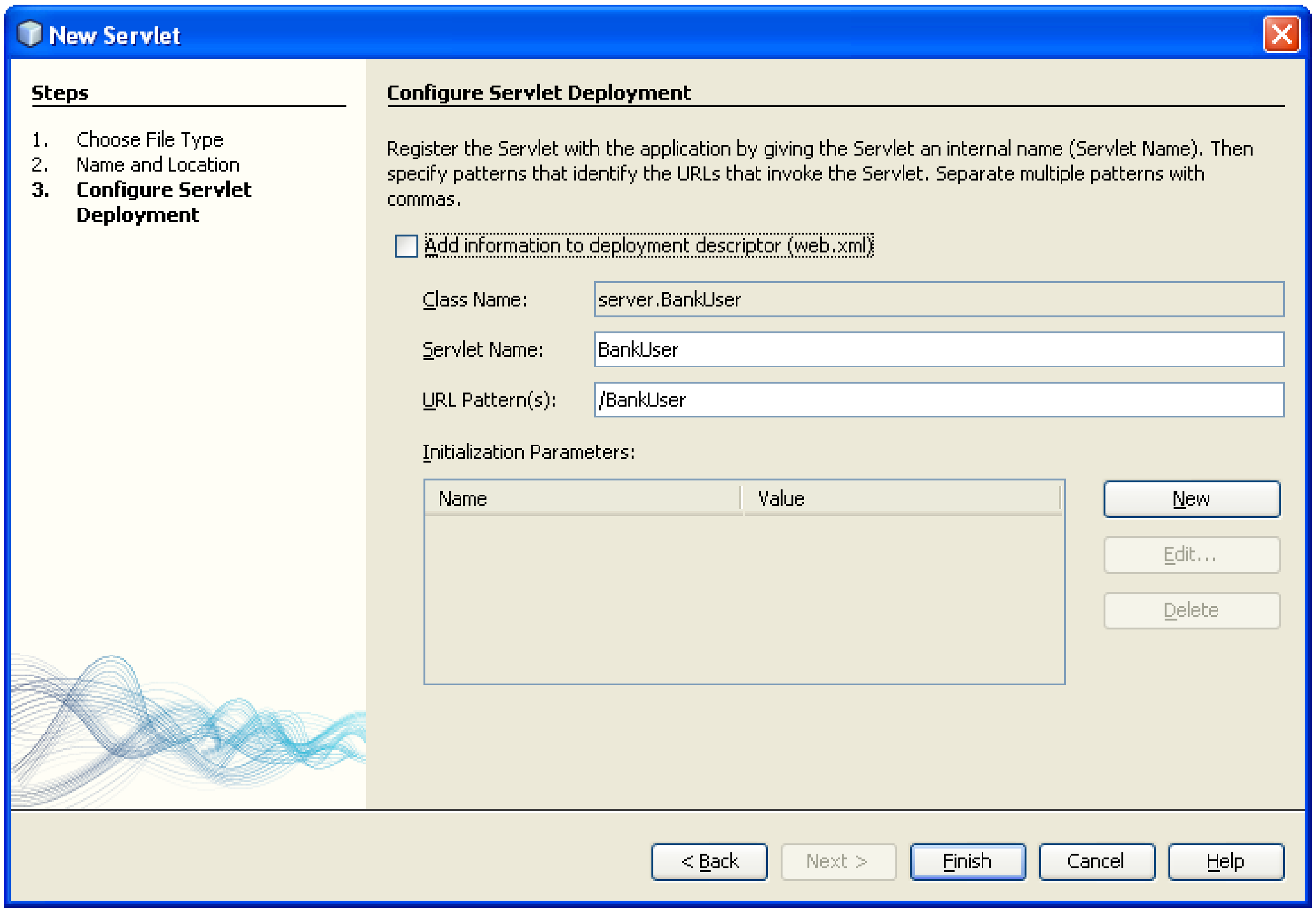
Class Name: BankUser

Project: WebBankingApp

Location: Source Packages

Package: server

Created File: 5 2010-2011\MY SLIDES\LESSON 9\Test\WebBankingApp\src\java\server\BankUser.java



Configure Servlet Deployment

Register the Servlet with the application by giving the Servlet an internal name (Servlet Name). Then specify patterns that identify the URLs that invoke the Servlet. Separate multiple patterns with commas.

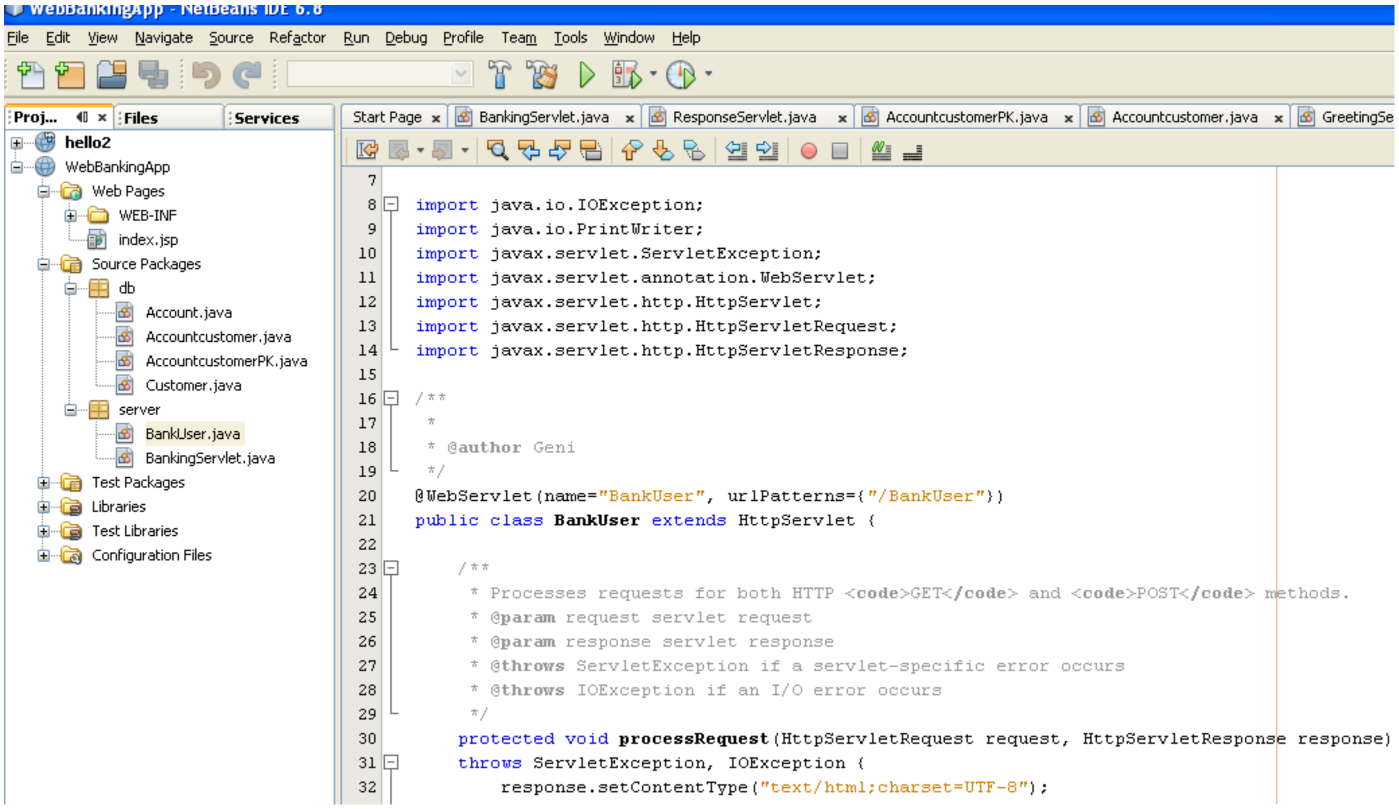
[ ] Add information to deployment descriptor (web.xml)

Class Name:	server.BankUser
Servlet Name:	BankUser
URL Pattern(s):	/BankUser

Initialization Parameters:

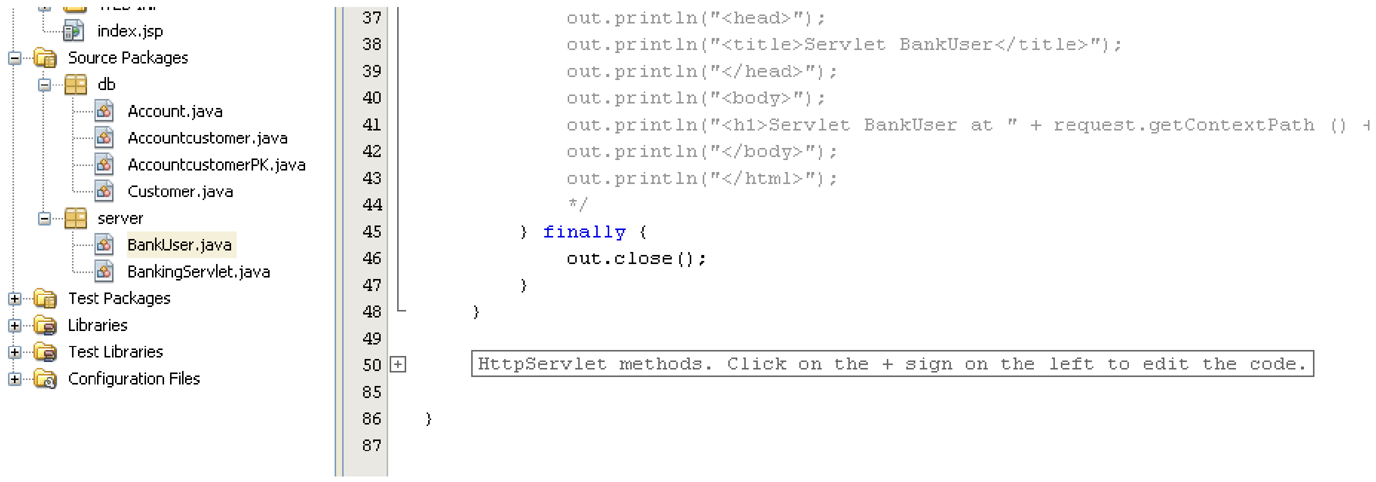
Name	Value

This is what you get:



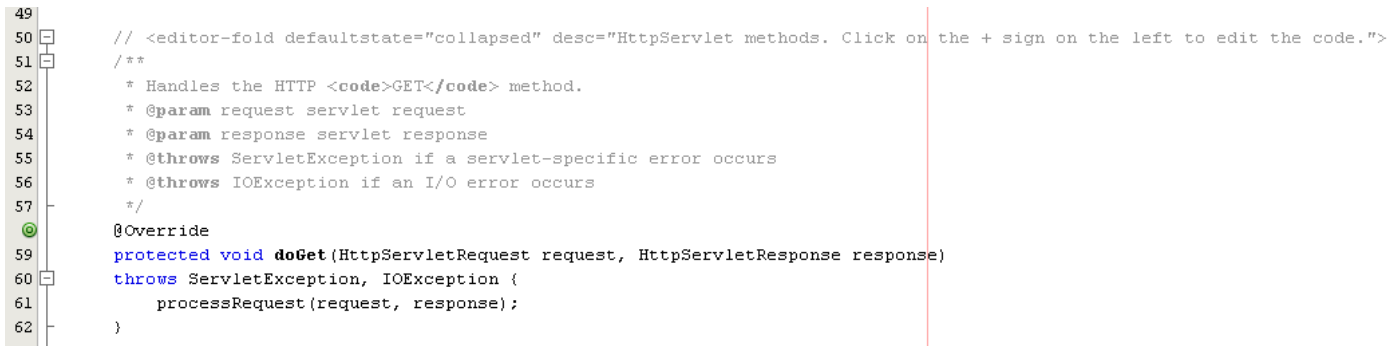
```
7
8   import java.io.IOException;
9   import java.io.PrintWriter;
10  import javax.servlet.ServletException;
11  import javax.servlet.annotation.WebServlet;
12  import javax.servlet.http.HttpServlet;
13  import javax.servlet.http.HttpServletRequest;
14  import javax.servlet.http.HttpServletResponse;
15
16  /**
17   *
18   * @author Geni
19   */
20  @WebServlet(name="BankUser", urlPatterns={"/BankUser"})
21  public class BankUser extends HttpServlet {
22
23      /**
24       * Processes requests for both HTTP <code>GET</code> and <code>POST</code> methods.
25       * @param request servlet request
26       * @param response servlet response
27       * @throws ServletException if a servlet-specific error occurs
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");
```

Now to the part of the code as follows:



```
37              out.println("<head>");
38              out.println("<title>Servlet BankUser</title>");
39              out.println("</head>");
40              out.println("<body>");
41              out.println("<h1>Servlet BankUser at " + request.getContextPath () +
42              out.println("</body>");
43              out.println("</html>");
44              */
45          } finally {
46              out.close();
47          }
48      }
49
50  +   HttpServlet methods. Click on the + sign on the left to edit the code.
85
86  }
87
```

Click on the Plus symbol (line 50 here)

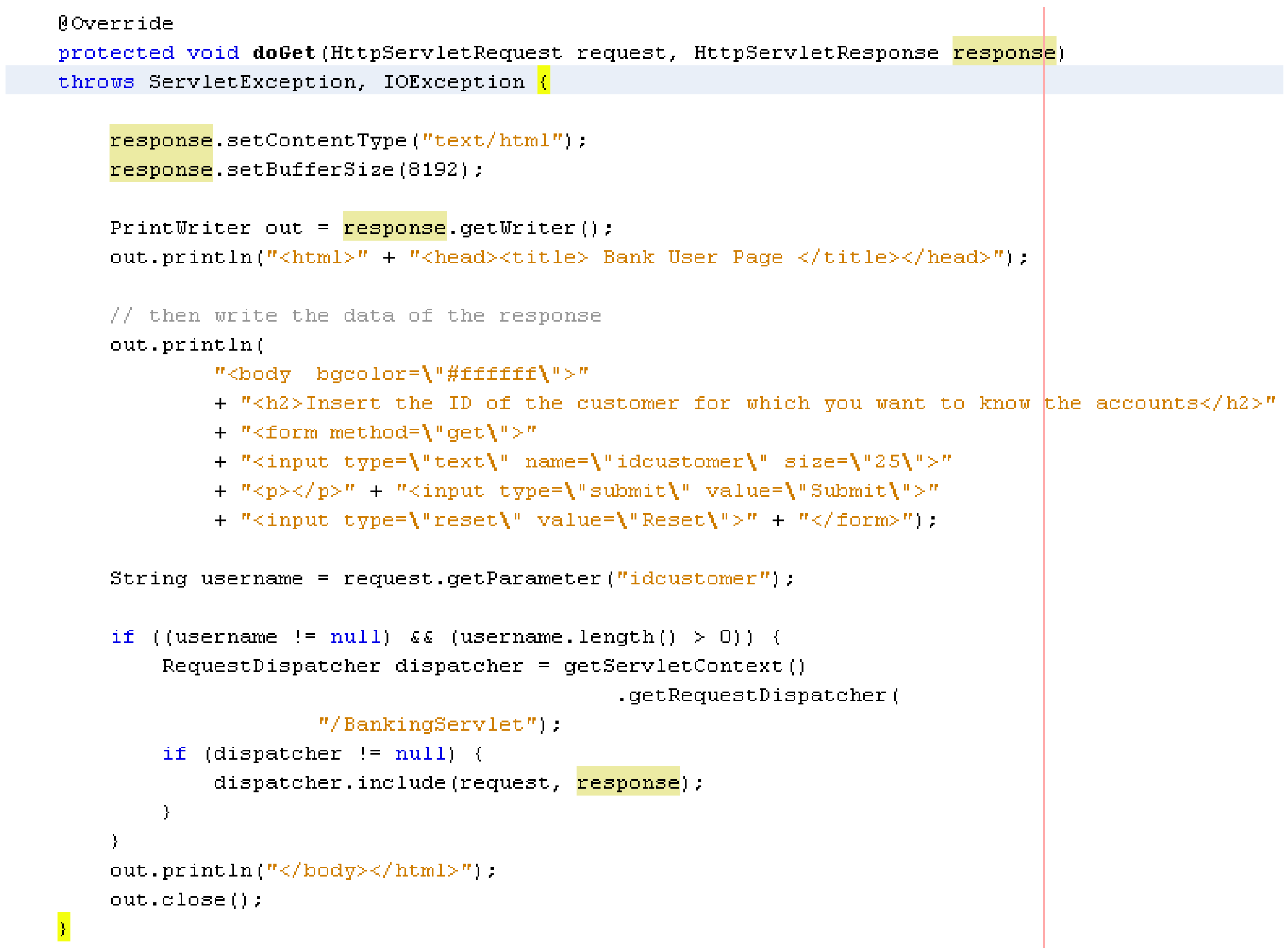


```
// <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the left to edit the code.">
/**
 * Handles the HTTP <code>GET</code> method.
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    processRequest(request, response);
}
```

We have to develop now the method doGet.

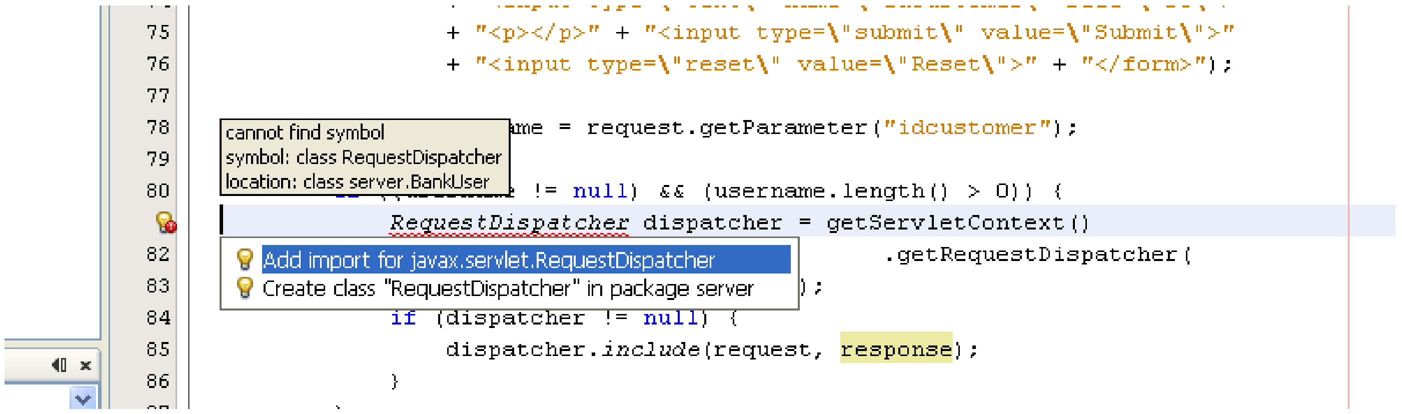
```
public class SomeServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

    // Use "request" to read incoming HTTP headers (e.g. cookies)
    // and HTML form data (e.g. data the user entered and submitted)

    // Use "response" to specify the HTTP response line and headers
    // (e.g. specifying the content type, setting cookies).

    PrintWriter out = response.getWriter();
    // Use "out" to send content to browser
  }
```

Add the following code within the body of doGet :



```java
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

    response.setContentType("text/html");
    response.setBufferSize(8192);

    PrintWriter out = response.getWriter();
    out.println("<html>" + "<head><title> Bank User Page </title></head>");

    // then write the data of the response
    out.println(
            "<body  bgcolor=\"#ffffff\">"
            + "<h2>Insert the ID of the customer for which you want to know the accounts</h2>"
            + "<form method=\"get\">"
            + "<input type=\"text\" name=\"idcustomer\" size=\"25\">"
            + "<p></p>" + "<input type=\"submit\" value=\"Submit\">"
            + "<input type=\"reset\" value=\"Reset\">" + "</form>");

    String username = request.getParameter("idcustomer");

    if ((username != null) && (username.length() > 0)) {
        RequestDispatcher dispatcher = getServletContext()
                                           .getRequestDispatcher(
                    "/BankingServlet");
        if (dispatcher != null) {
            dispatcher.include(request, response);
        }
    }
    out.println("</body></html>");
    out.close();
}
```

Fix the imports:

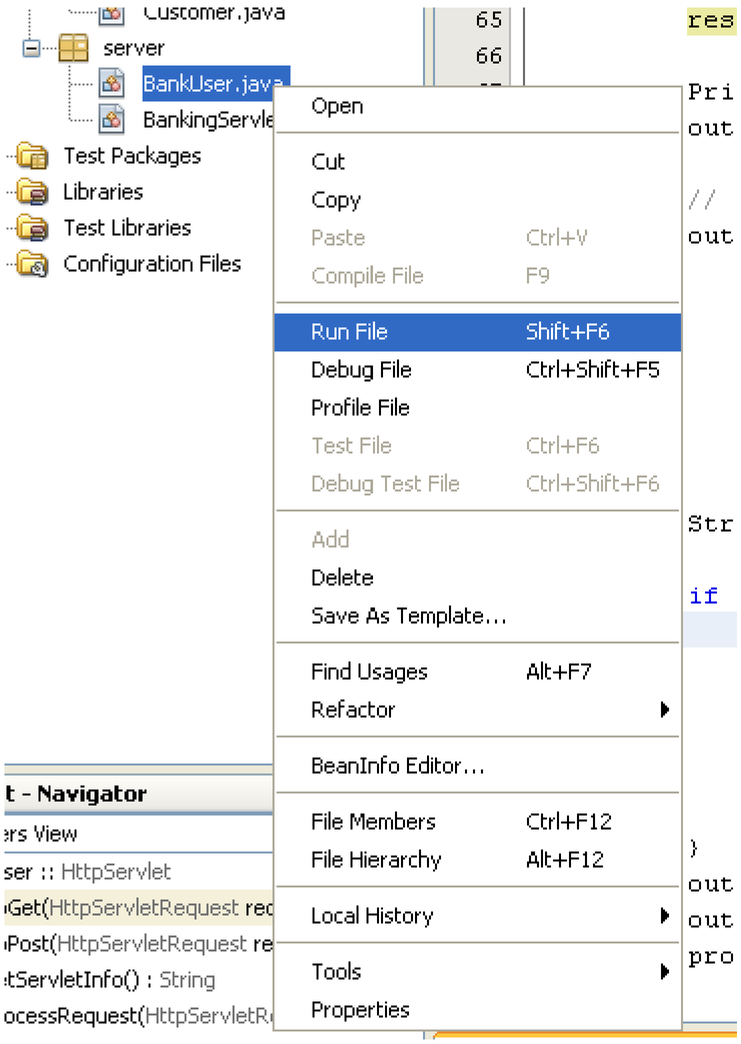


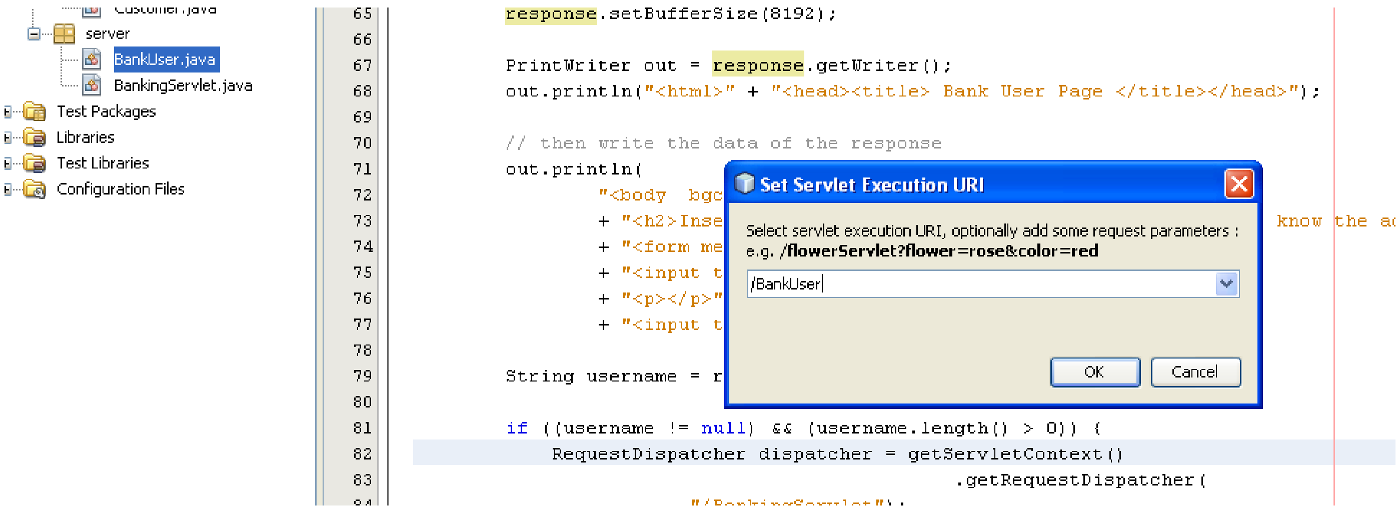
```java
            + "<p></p>" + "<input type=\"submit\" value=\"Submit\">"
            + "<input type=\"reset\" value=\"Reset\">" + "</form>");

    String username = request.getParameter("idcustomer");

    if ((username != null) && (username.length() > 0)) {
        RequestDispatcher dispatcher = getServletContext()
                                           .getRequestDispatcher(
                    "/BankingServlet");
        if (dispatcher != null) {
            dispatcher.include(request, response);
        }
```

```
cannot find symbol
symbol: class RequestDispatcher
location: class server.BankUser
```

```
Add import for javax.servlet.RequestDispatcher
Create class "RequestDispatcher" in package server
```

If you run the file you will see that we have developed the following page:



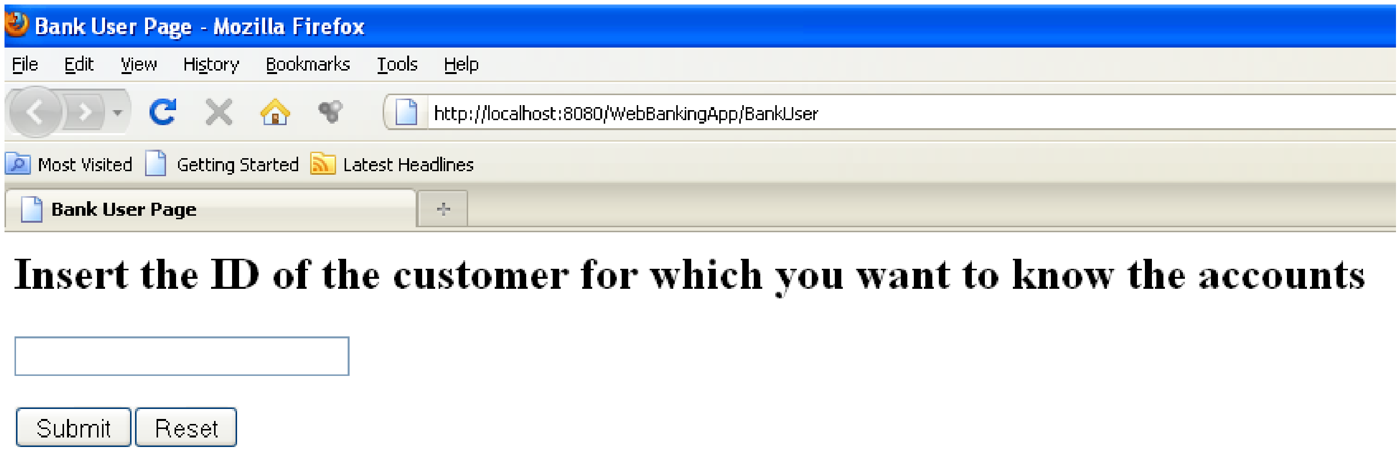


```
        response.setBufferSize(8192);

        PrintWriter out = response.getWriter();
        out.println("<html>" + "<head><title> Bank User Page </title></head>");

        // then write the data of the response
        out.println(
                "<body  bgc
                + "<h2>Inse                                          know the ac
                + "<form me
                + "<input t
                + "<p></p>"
                + "<input t

        String username = r

        if ((username != null) && (username.length() > 0)) {
            RequestDispatcher dispatcher = getServletContext()
                                            .getRequestDispatcher(
                    "/BankingServlet");
```

Set Servlet Execution URI

Select servlet execution URI, optionally add some request parameters :
e.g. **/flowerServlet?flower=rose&color=red**

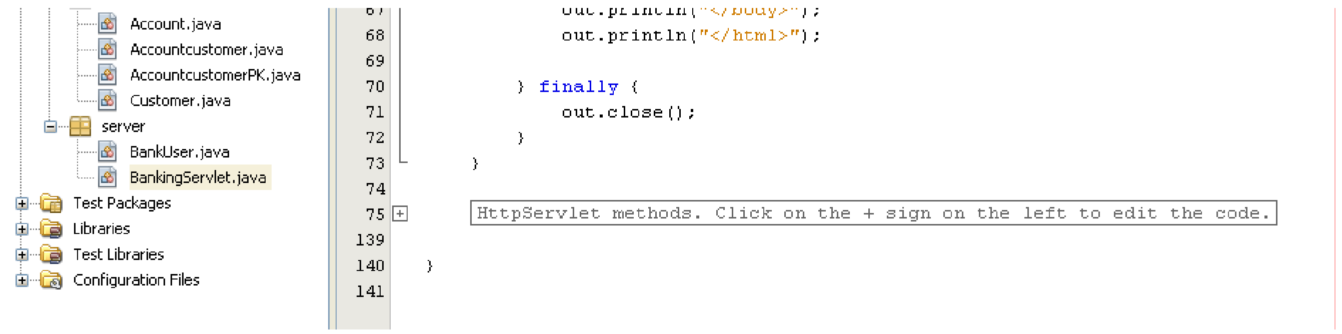
/BankUser


Bank User Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebBankingApp/BankUser
Most Visited Getting Started Latest Headlines
Bank User Page

Insert the ID of the customer for which you want to know the accounts

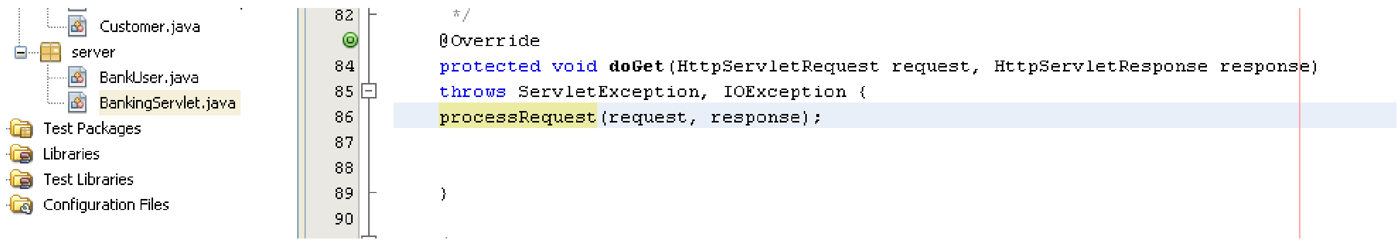
Now we have to develop the response Servlet that will perform the query on the database:

Go the BankingServlet and click the following “Plus” as shown:



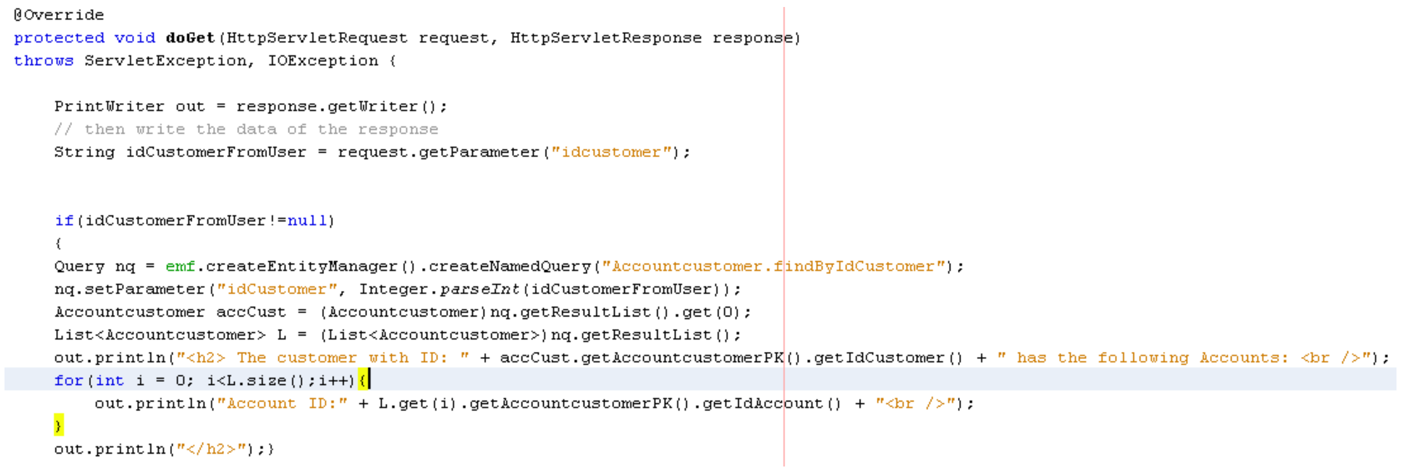
```
                out.println("</body>");
                out.println("</html>");

            } finally {
                out.close();
            }
        }

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

    }
```

You see the following when you expand the methods:



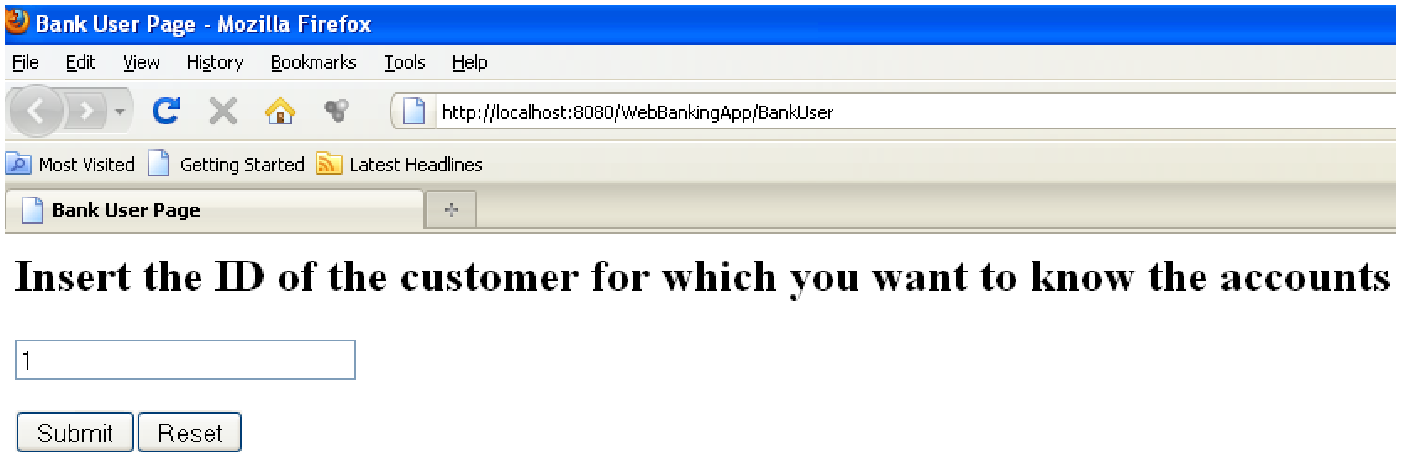
```
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);


    }
```

Now write the following code within the body of the method doGet:



```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

    PrintWriter out = response.getWriter();
    // then write the data of the response
    String idCustomerFromUser = request.getParameter("idcustomer");


    if(idCustomerFromUser!=null)
    {
    Query nq = emf.createEntityManager().createNamedQuery("Accountcustomer.findByIdCustomer");
    nq.setParameter("idCustomer", Integer.parseInt(idCustomerFromUser));
    Accountcustomer accCust = (Accountcustomer)nq.getResultList().get(0);
    List<Accountcustomer> L = (List<Accountcustomer>)nq.getResultList();
    out.println("<h2> The customer with ID: " + accCust.getAccountcustomerPK().getIdCustomer() + " has the following Accounts: <br />");
    for(int i = 0; i<L.size();i++){
        out.println("Account ID:" + L.get(i).getAccountcustomerPK().getIdAccount() + "<br />");
    }
    out.println("</h2>");}
```

If you run the file BankUser again you can send a request to the BankingServlet:

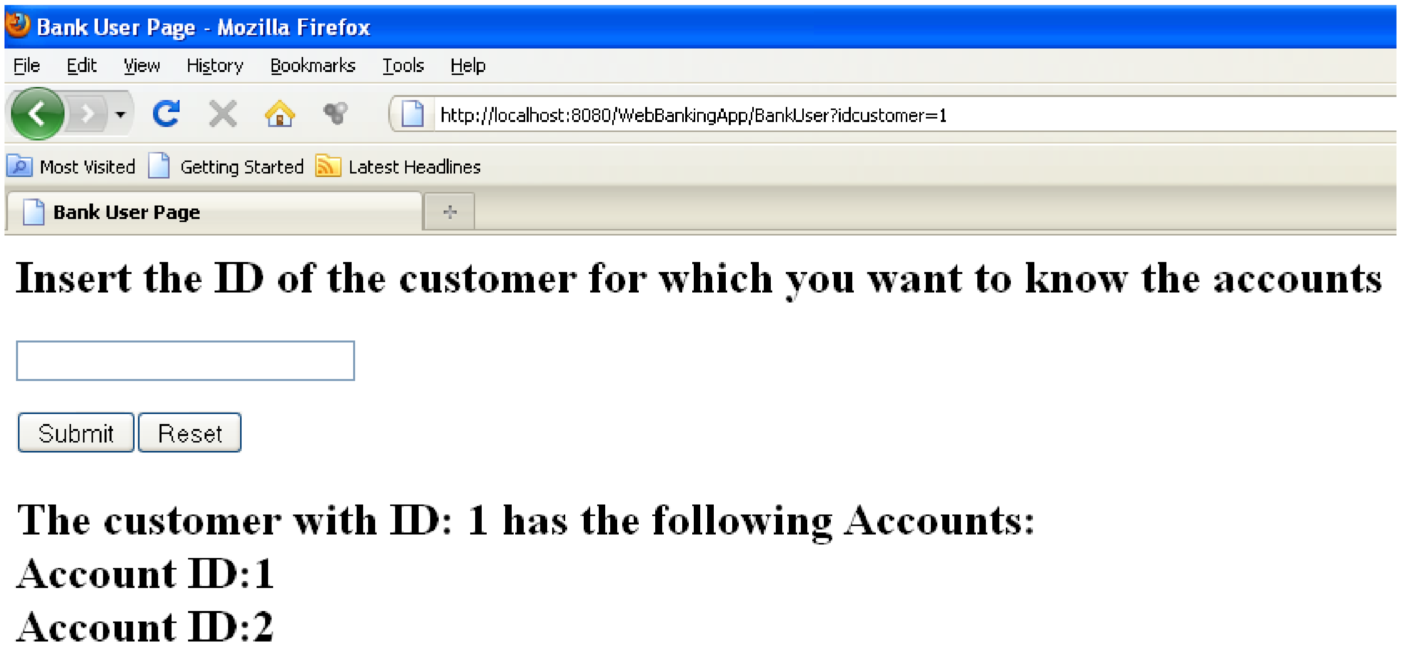


Bank User Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebBankingApp/BankUser
Most Visited Getting Started Latest Headlines
Bank User Page

Insert the ID of the customer for which you want to know the accounts

1

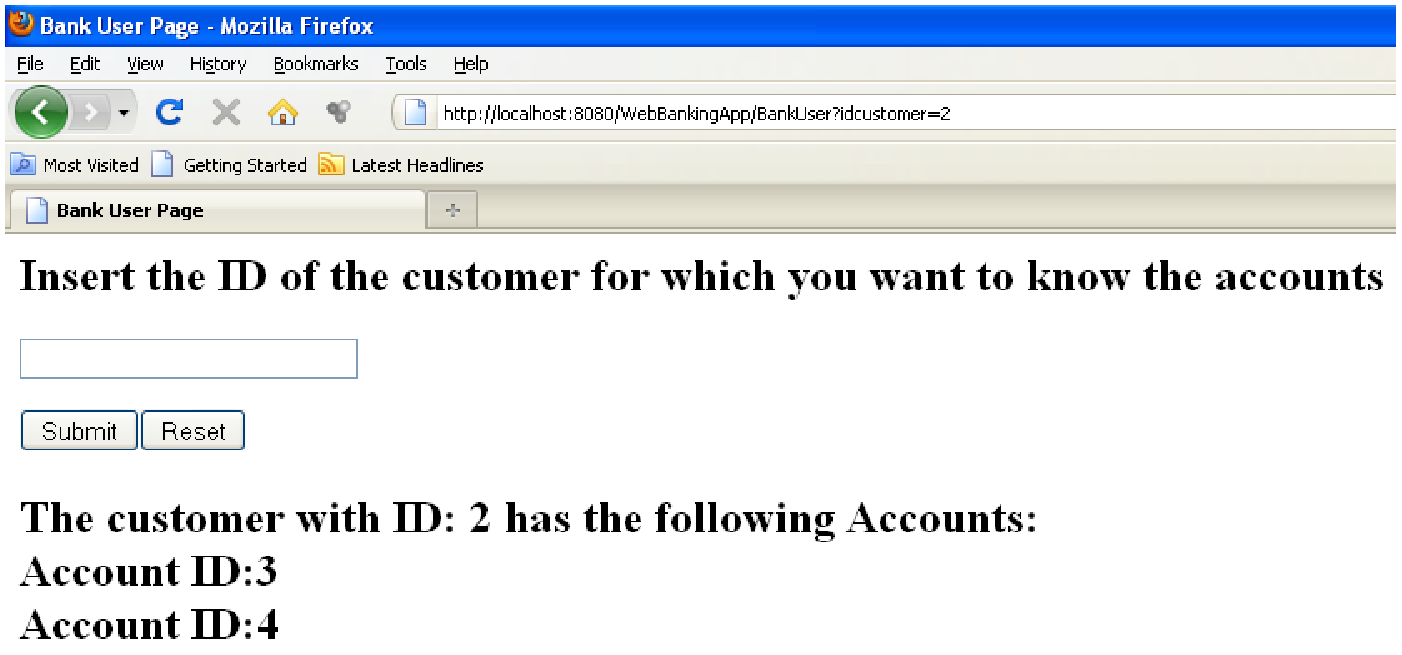
Press “Submit” and you will see the following on the page:


Bank User Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebBankingApp/BankUser?idcustomer=1
Most Visited Getting Started Latest Headlines
Bank User Page

Insert the ID of the customer for which you want to know the accounts

Submit Reset

The customer with ID: 1 has the following Accounts:
Account ID:1
Account ID:2


Bank User Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebBankingApp/BankUser?idcustomer=2
Most Visited Getting Started Latest Headlines
Bank User Page

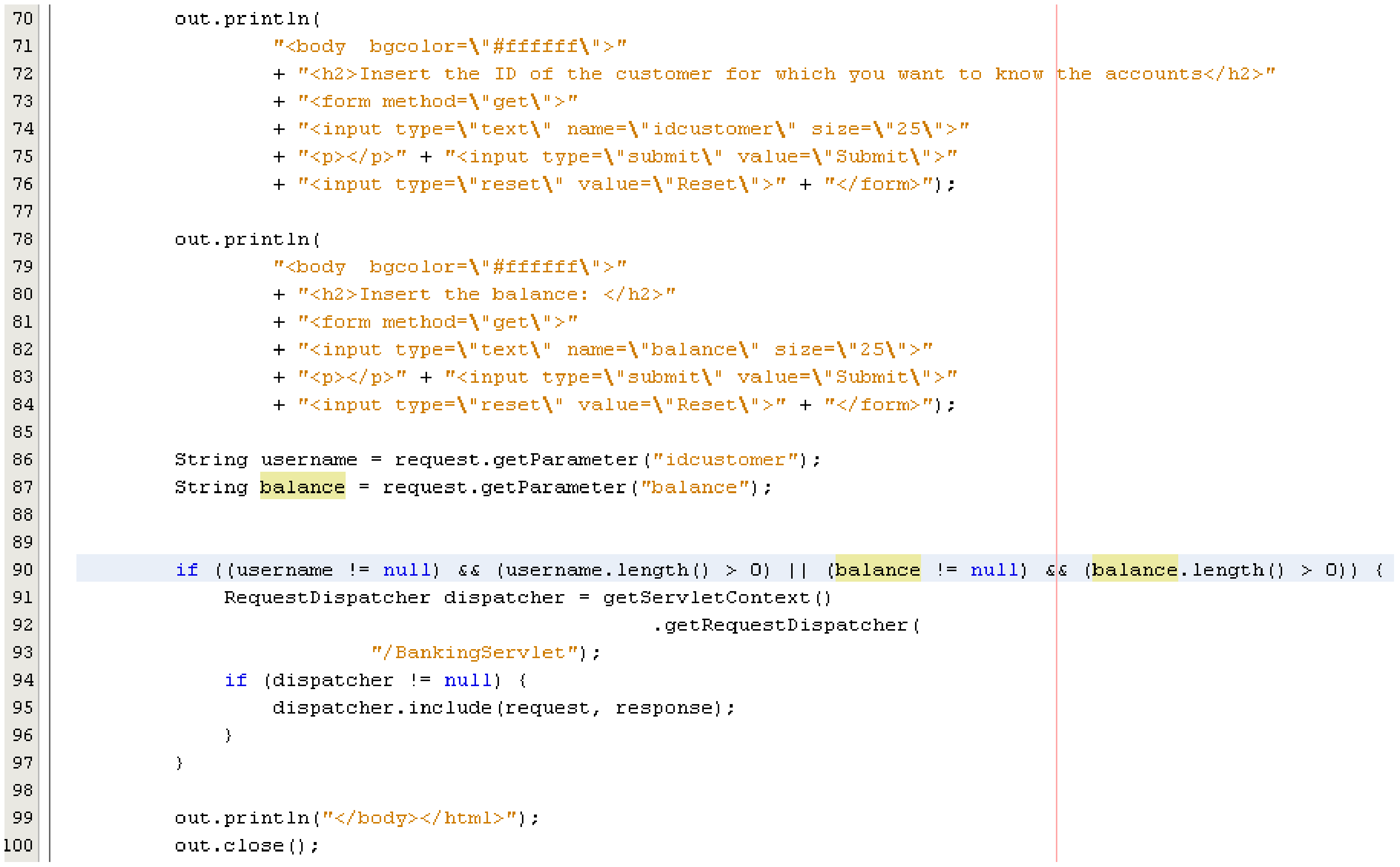
Insert the ID of the customer for which you want to know the accounts

Submit Reset

The customer with ID: 2 has the following Accounts:
Account ID:3
Account ID:4

Now we develop a method that will show all the accounts with balance greater than a certain value inserted by the user.

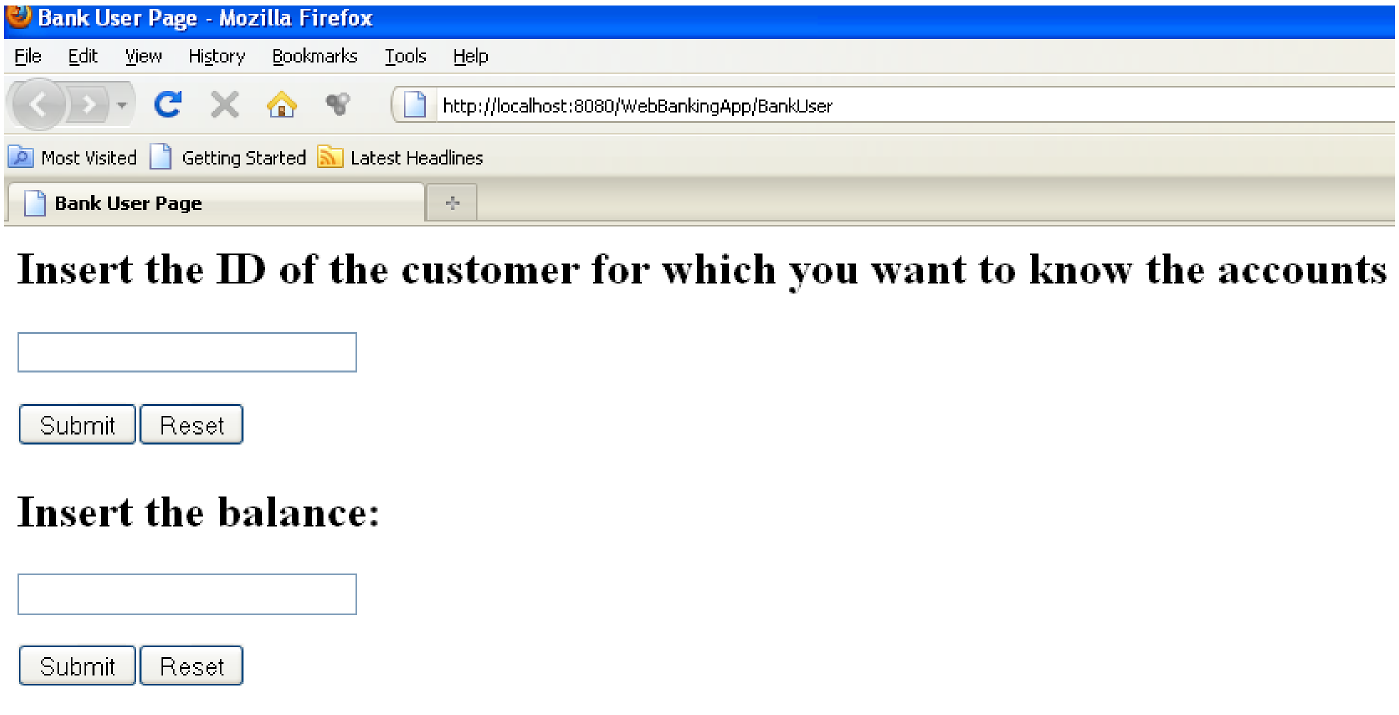
Change the code as follows:



```
 70         out.println(
 71                 "<body  bgcolor=\"#ffffff\">"
 72                 + "<h2>Insert the ID of the customer for which you want to know the accounts</h2>"
 73                 + "<form method=\"get\">"
 74                 + "<input type=\"text\" name=\"idcustomer\" size=\"25\">"
 75                 + "<p></p>" + "<input type=\"submit\" value=\"Submit\">"
 76                 + "<input type=\"reset\" value=\"Reset\">" + "</form>");
 77 
 78         out.println(
 79                 "<body  bgcolor=\"#ffffff\">"
 80                 + "<h2>Insert the balance: </h2>"
 81                 + "<form method=\"get\">"
 82                 + "<input type=\"text\" name=\"balance\" size=\"25\">"
 83                 + "<p></p>" + "<input type=\"submit\" value=\"Submit\">"
 84                 + "<input type=\"reset\" value=\"Reset\">" + "</form>");
 85 
 86         String username = request.getParameter("idcustomer");
 87         String balance = request.getParameter("balance");
 88 
 89 
 90         if ((username != null) && (username.length() > 0) || (balance != null) && (balance.length() > 0)) {
 91             RequestDispatcher dispatcher = getServletContext()
 92                                                .getRequestDispatcher(
 93                         "/BankingServlet");
 94             if (dispatcher != null) {
 95                 dispatcher.include(request, response);
 96             }
 97         }
 98 
 99         out.println("</body></html>");
100         out.close();
```

Note the changes in the lines: 78-84, line 87, line 90.

If you run the file BankUser you will see the following:



Bank User Page - Mozilla Firefox

http://localhost:8080/WebBankingApp/BankUser

Insert the ID of the customer for which you want to know the accounts

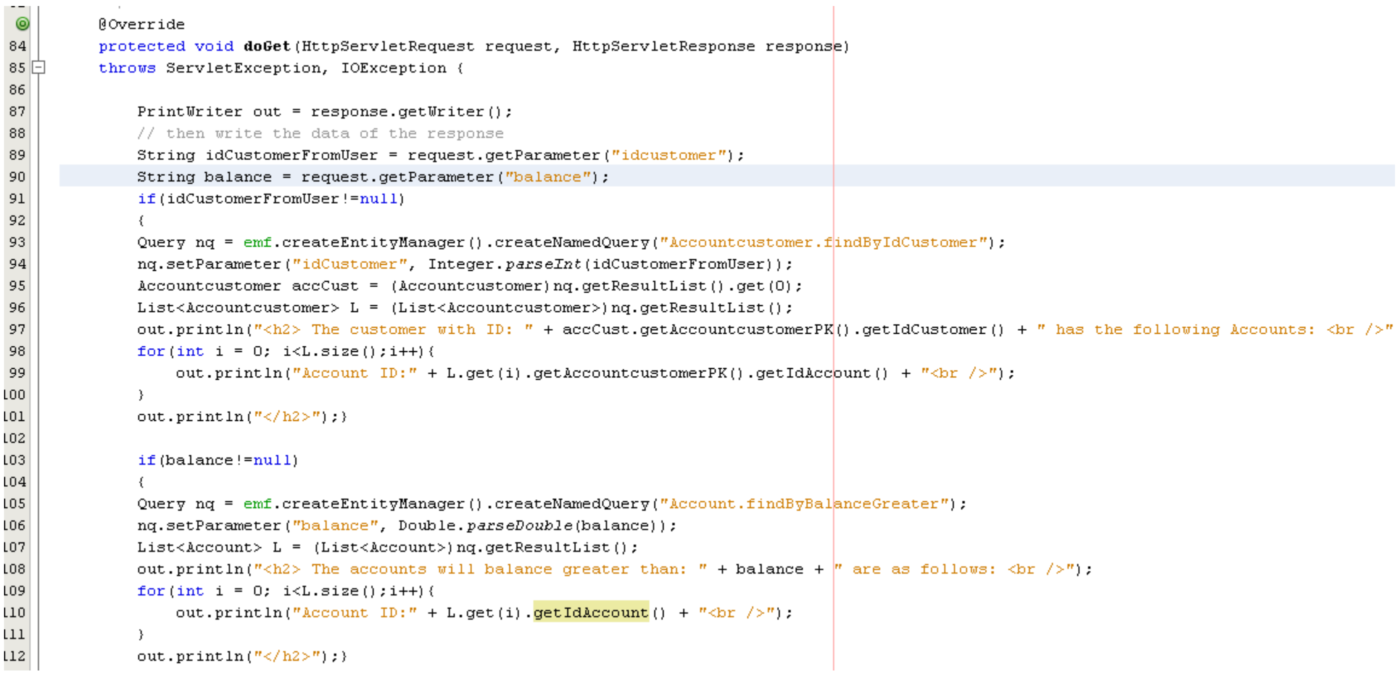
Submit Reset

Insert the balance:

Submit Reset

Now we need to extend the BankingServlet.

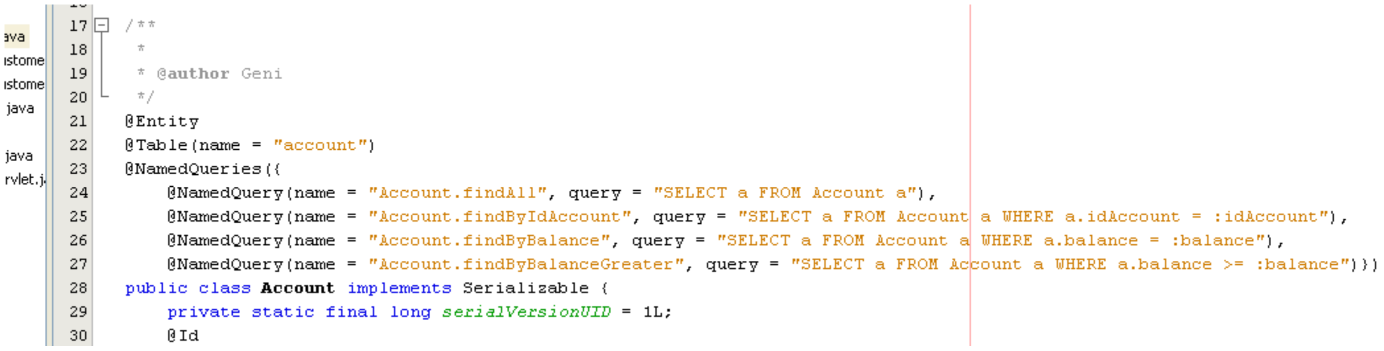
Add the following code:



```
        @Override
 84     protected void doGet(HttpServletRequest request, HttpServletResponse response)
 85     throws ServletException, IOException {
 86
 87         PrintWriter out = response.getWriter();
 88         // then write the data of the response
 89         String idCustomerFromUser = request.getParameter("idcustomer");
 90         String balance = request.getParameter("balance");
 91         if(idCustomerFromUser!=null)
 92         {
 93         Query nq = emf.createEntityManager().createNamedQuery("Accountcustomer.findByIdCustomer");
 94         nq.setParameter("idCustomer", Integer.parseInt(idCustomerFromUser));
 95         Accountcustomer accCust = (Accountcustomer)nq.getResultList().get(0);
 96         List<Accountcustomer> L = (List<Accountcustomer>)nq.getResultList();
 97         out.println("<h2> The customer with ID: " + accCust.getAccountcustomerPK().getIdCustomer() + " has the following Accounts: <br />"
 98         for(int i = 0; i<L.size();i++){
 99             out.println("Account ID:" + L.get(i).getAccountcustomerPK().getIdAccount() + "<br />");
100         }
101         out.println("</h2>");}
102
103         if(balance!=null)
104         {
105         Query nq = emf.createEntityManager().createNamedQuery("Account.findByBalanceGreater");
106         nq.setParameter("balance", Double.parseDouble(balance));
107         List<Account> L = (List<Account>)nq.getResultList();
108         out.println("<h2> The accounts will balance greater than: " + balance + " are as follows: <br />");
109         for(int i = 0; i<L.size();i++){
110             out.println("Account ID:" + L.get(i).getIdAccount() + "<br />");
111         }
112         out.println("</h2>");}
```

Note the changes in the line 90, 103-112.

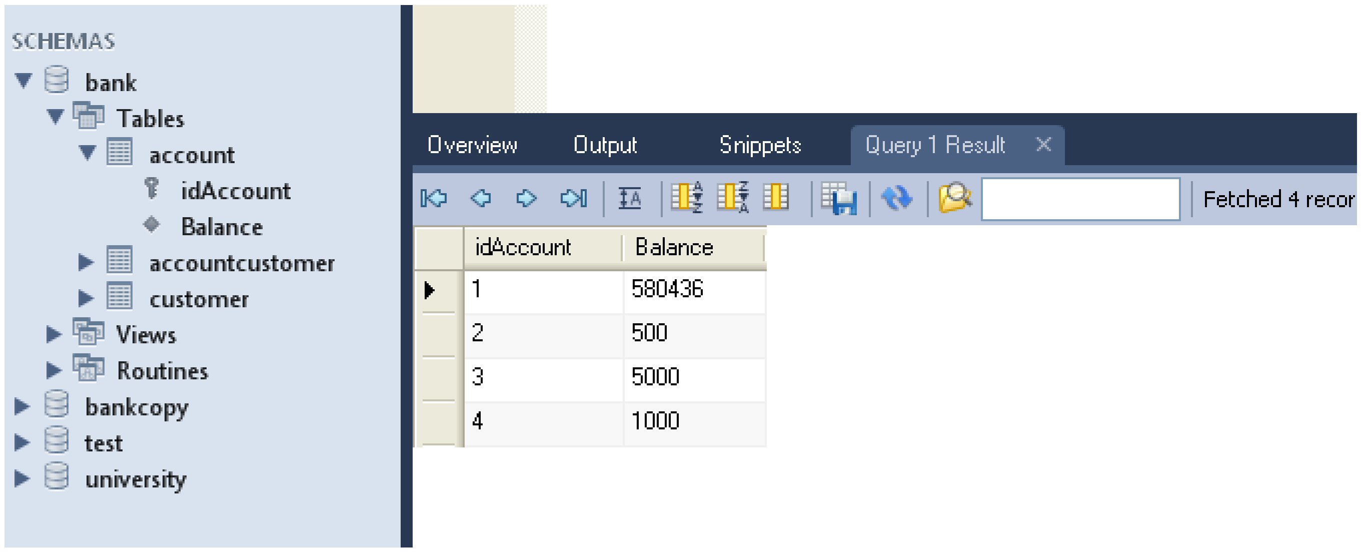
We also need to change the class Account as follows:



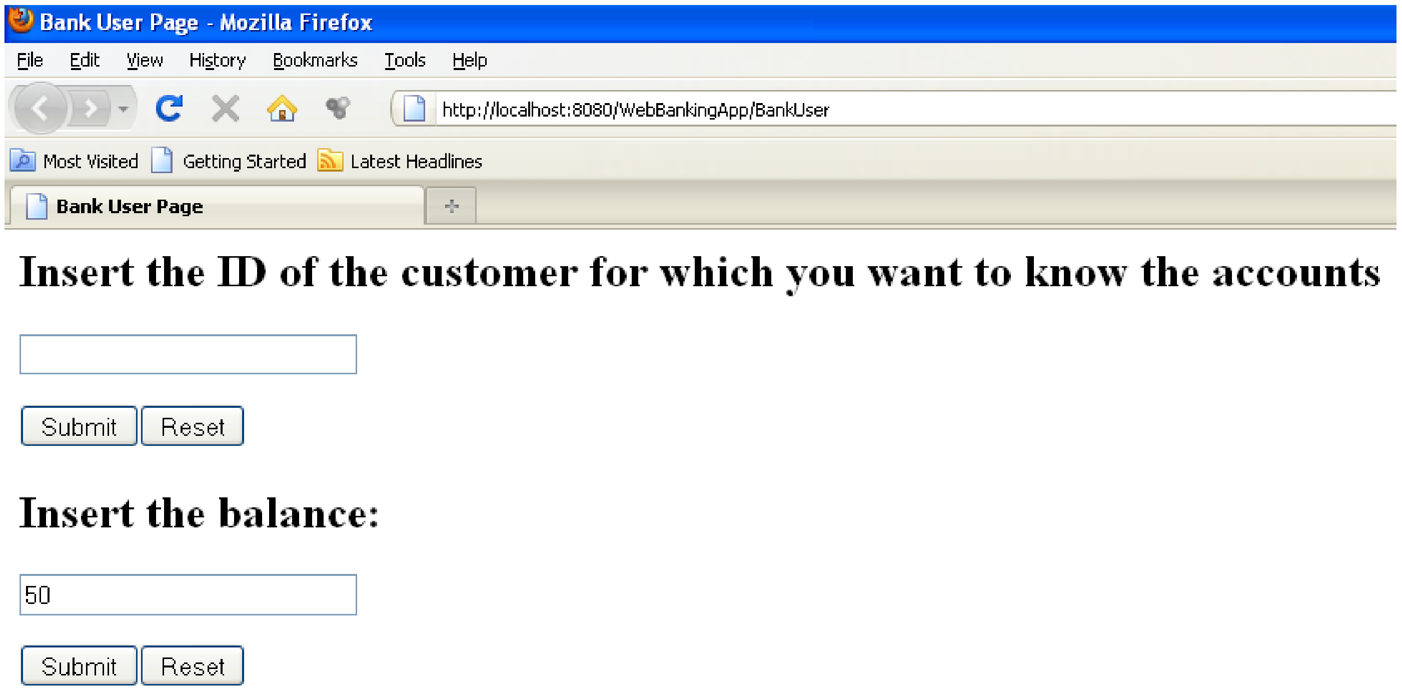
```
17  /**
18   *
19   * @author Geni
20   */
21  @Entity
22  @Table(name = "account")
23  @NamedQueries({
24      @NamedQuery(name = "Account.findAll", query = "SELECT a FROM Account a"),
25      @NamedQuery(name = "Account.findByIdAccount", query = "SELECT a FROM Account a WHERE a.idAccount = :idAccount"),
26      @NamedQuery(name = "Account.findByBalance", query = "SELECT a FROM Account a WHERE a.balance = :balance"),
27      @NamedQuery(name = "Account.findByBalanceGreater", query = "SELECT a FROM Account a WHERE a.balance >= :balance")})
28  public class Account implements Serializable {
29      private static final long serialVersionUID = 1L;
30      @Id
```

Note the change in line 27 where we define a new SQL query to get the accounts with balance greater than a certain parameter

Now we can execute BankUser. The status of the database is the following:



idAccount	Balance
1	580436
2	500
3	5000
4	1000

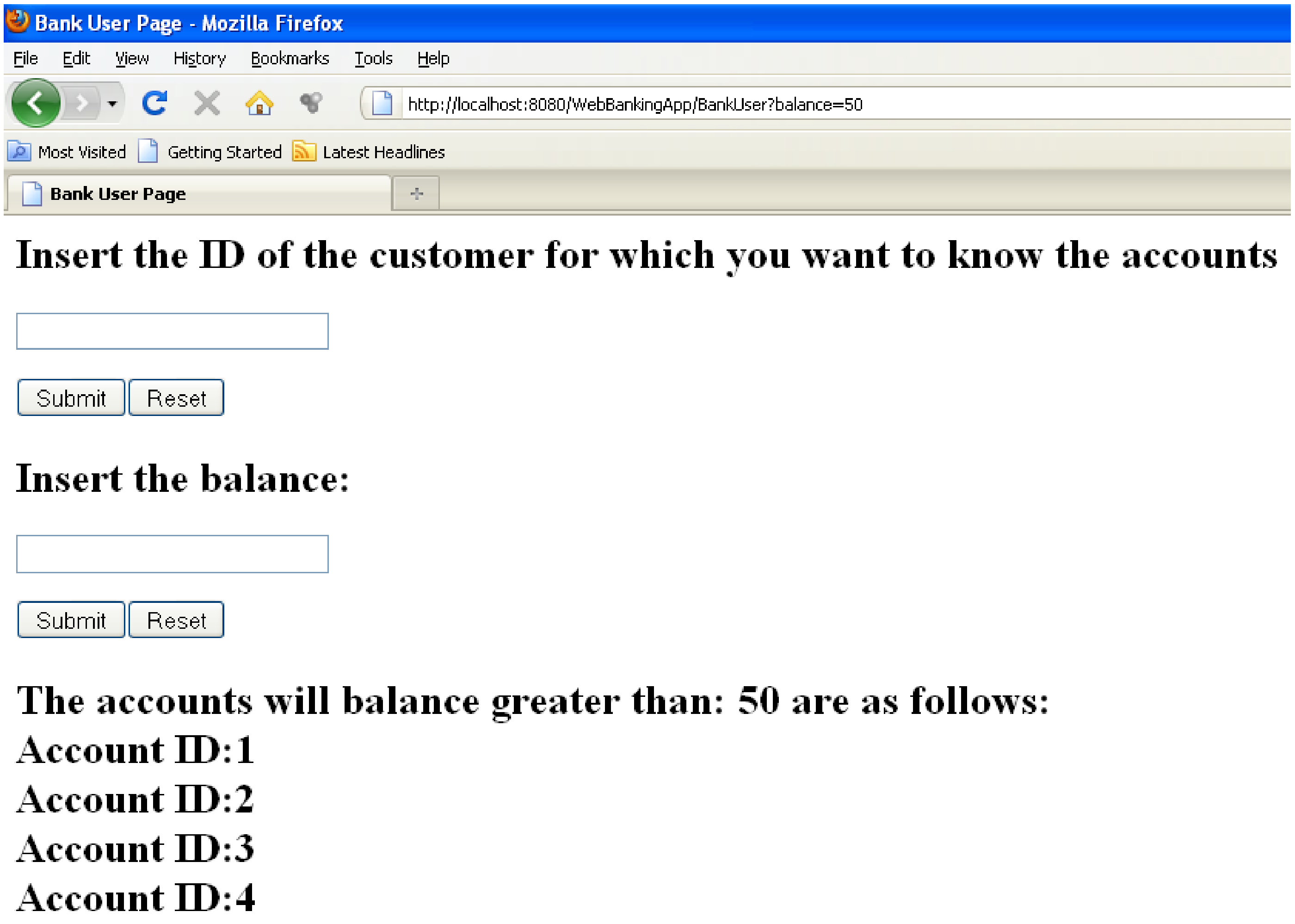

Bank User Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebBankingApp/BankUser
Most Visited
Getting Started
Latest Headlines
Bank User Page

Insert the ID of the customer for which you want to know the accounts

Insert the balance:

50

Press Submit and you will see:

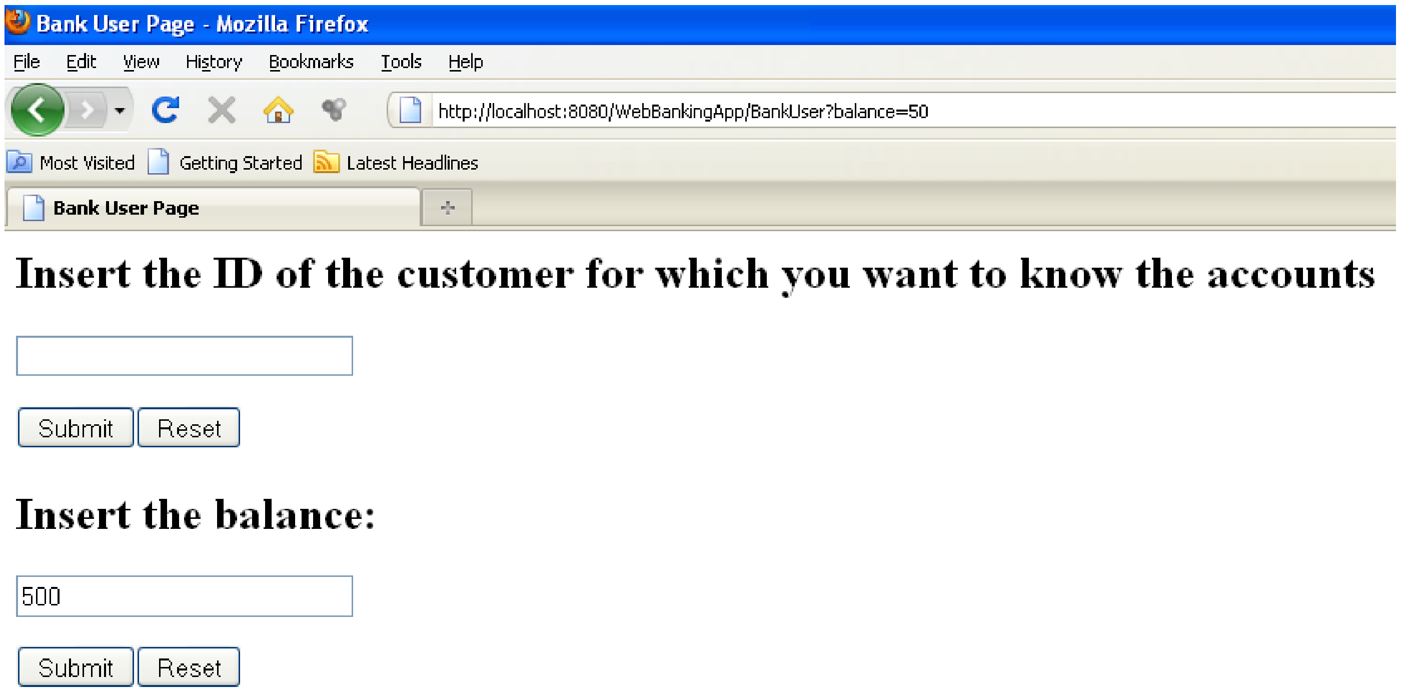

Bank User Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebBankingApp/BankUser?balance=50
Most Visited Getting Started Latest Headlines
Bank User Page
Submit Reset
Submit Reset

Insert the ID of the customer for which you want to know the accounts

Insert the balance:

The accounts will balance greater than: 50 are as follows:
Account ID:1
Account ID:2
Account ID:3
Account ID:4

Then try with 500:


Bank User Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebBankingApp/BankUser?balance=50
Most Visited Getting Started Latest Headlines
Bank User Page

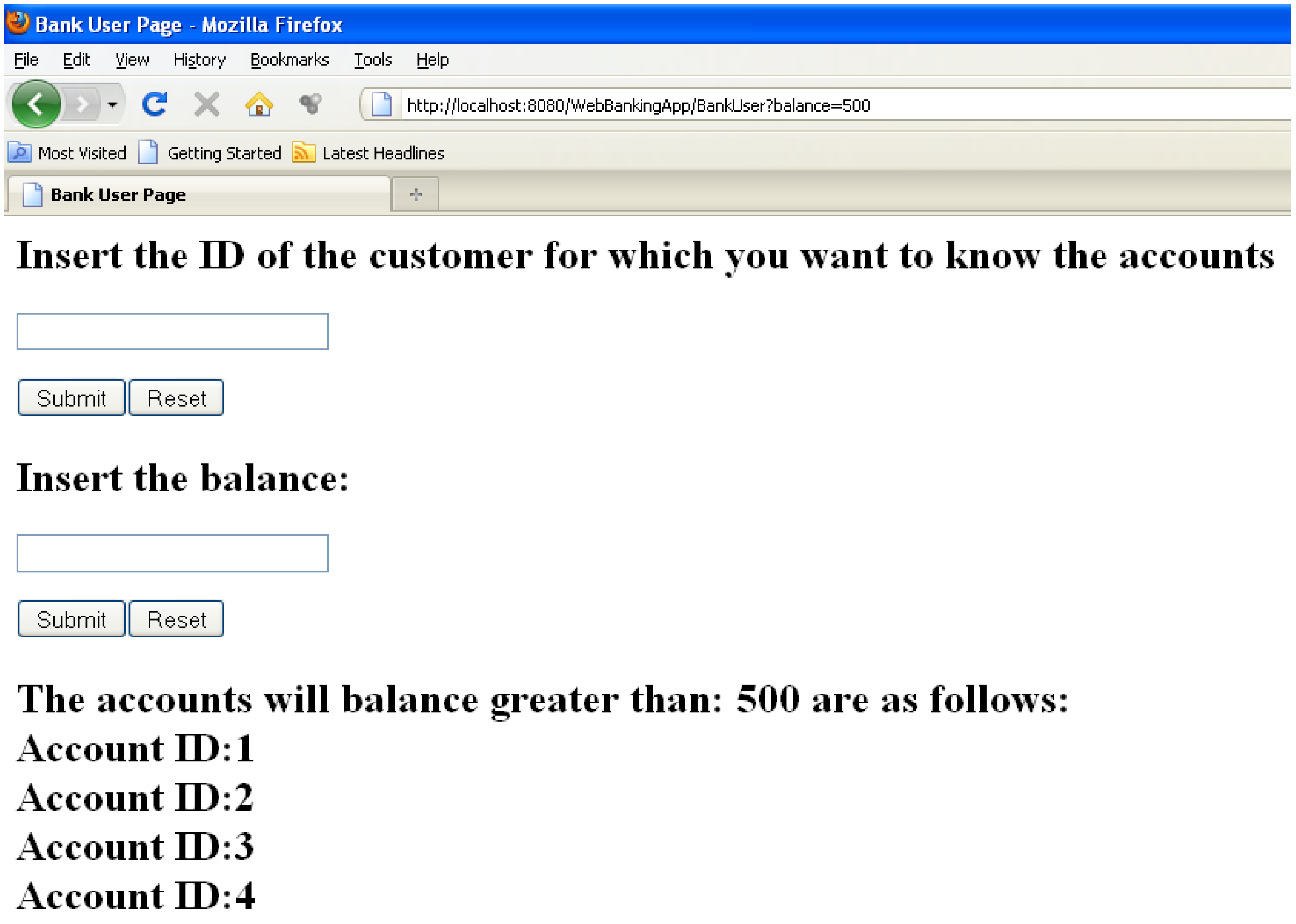
Insert the ID of the customer for which you want to know the accounts

Submit Reset

Insert the balance:

500

Submit Reset


Bank User Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebBankingApp/BankUser?balance=500
Most Visited Getting Started Latest Headlines
Bank User Page

Insert the ID of the customer for which you want to know the accounts

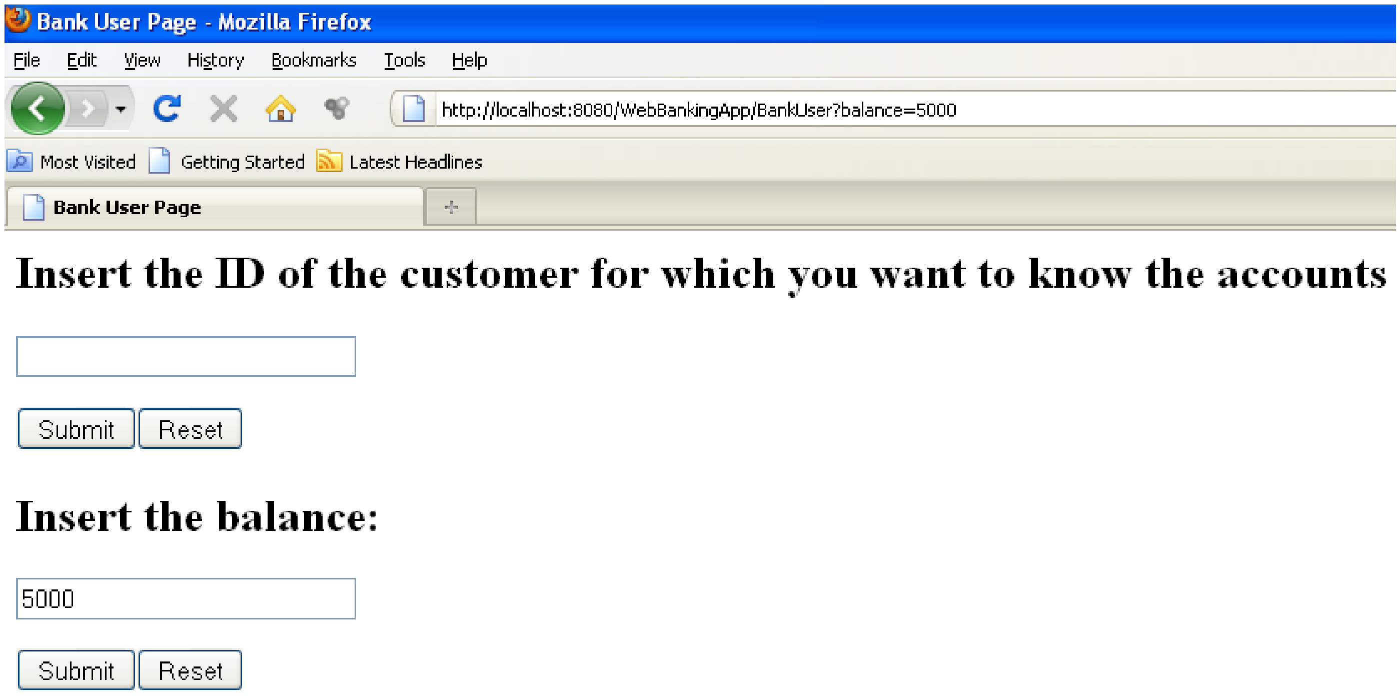
Submit Reset

Insert the balance:

Submit Reset

The accounts will balance greater than: 500 are as follows:
Account ID:1
Account ID:2
Account ID:3
Account ID:4

Try with 5000:


Bank User Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Most Visited Getting Started Latest Headlines
Bank User Page
Submit
Reset
Submit
Reset

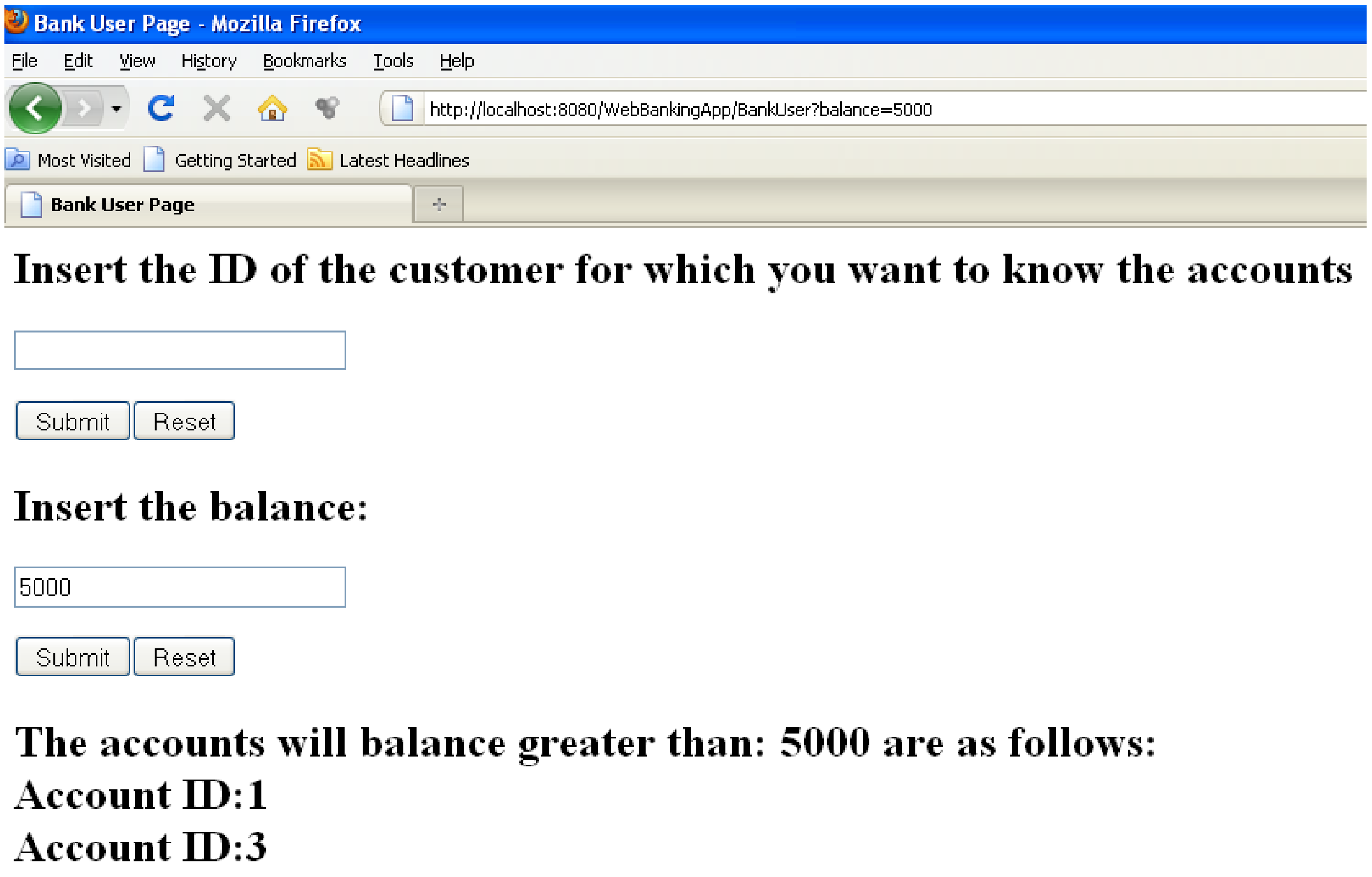
http://localhost:8080/WebBankingApp/BankUser?balance=5000

Insert the ID of the customer for which you want to know the accounts

Insert the balance:

5000

Press “Submit”:


Bank User Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Most Visited Getting Started Latest Headlines
Bank User Page
Submit Reset
Submit Reset

http://localhost:8080/WebBankingApp/BankUser?balance=5000

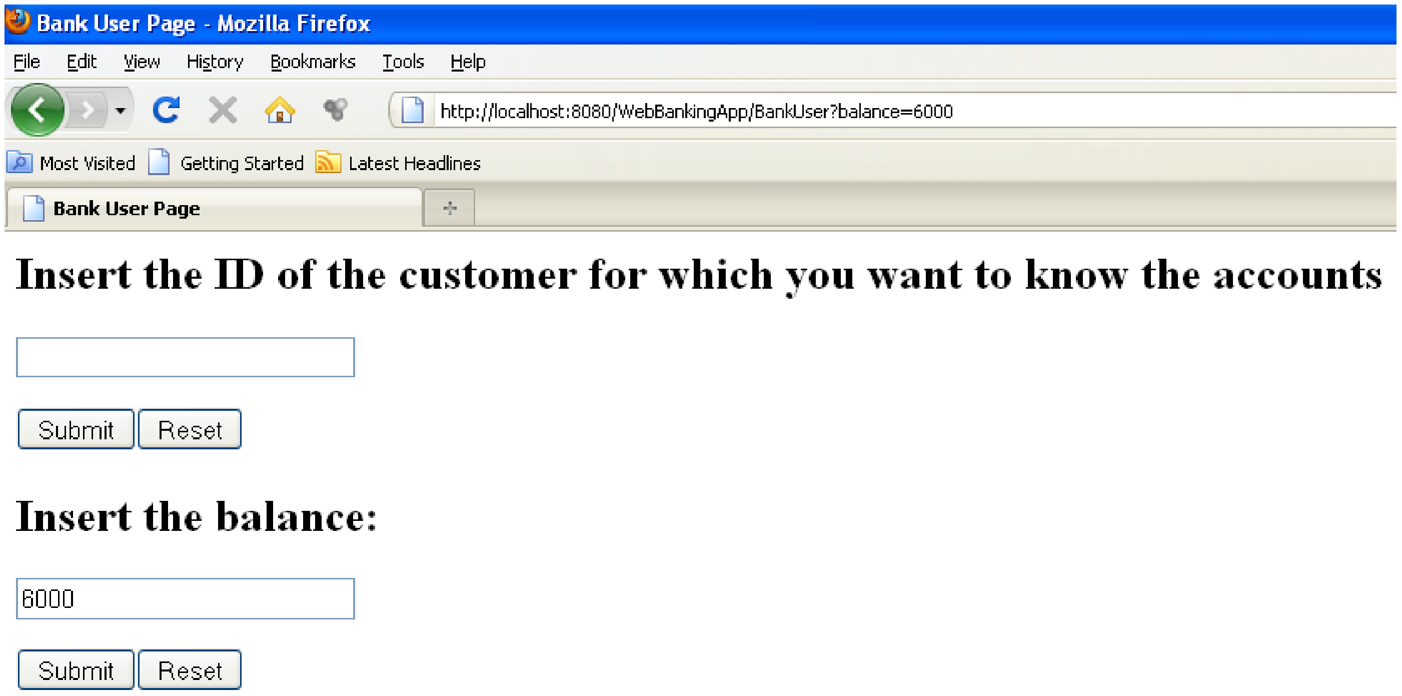
Insert the ID of the customer for which you want to know the accounts

Insert the balance:

5000

The accounts will balance greater than: 5000 are as follows:
Account ID:1
Account ID:3

We try with 6000:

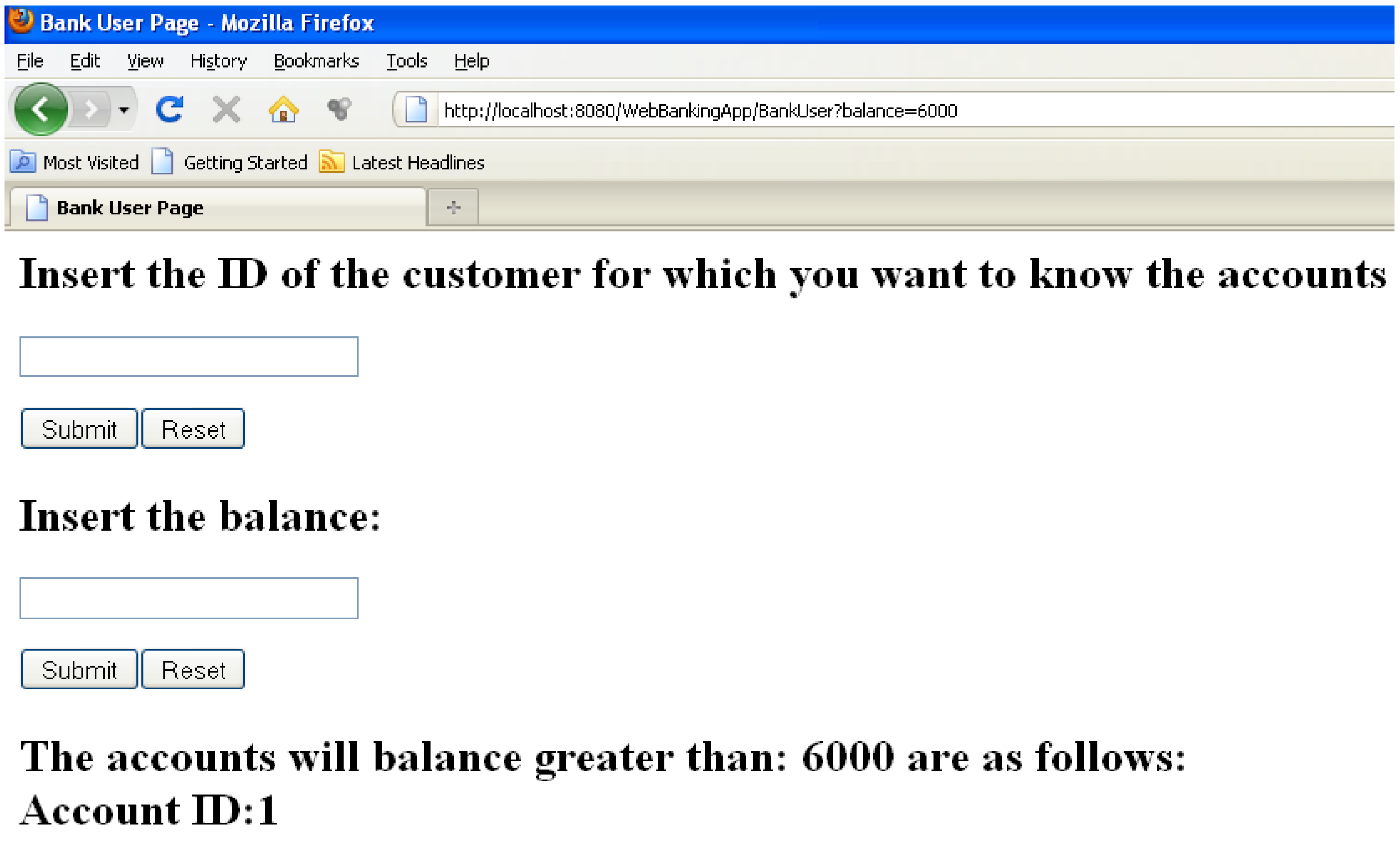

Bank User Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Most Visited Getting Started Latest Headlines
Bank User Page
Submit Reset
Submit Reset

http://localhost:8080/WebBankingApp/BankUser?balance=6000

Insert the ID of the customer for which you want to know the accounts

Insert the balance:

6000


Bank User Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebBankingApp/BankUser?balance=6000
Most Visited Getting Started Latest Headlines
Bank User Page

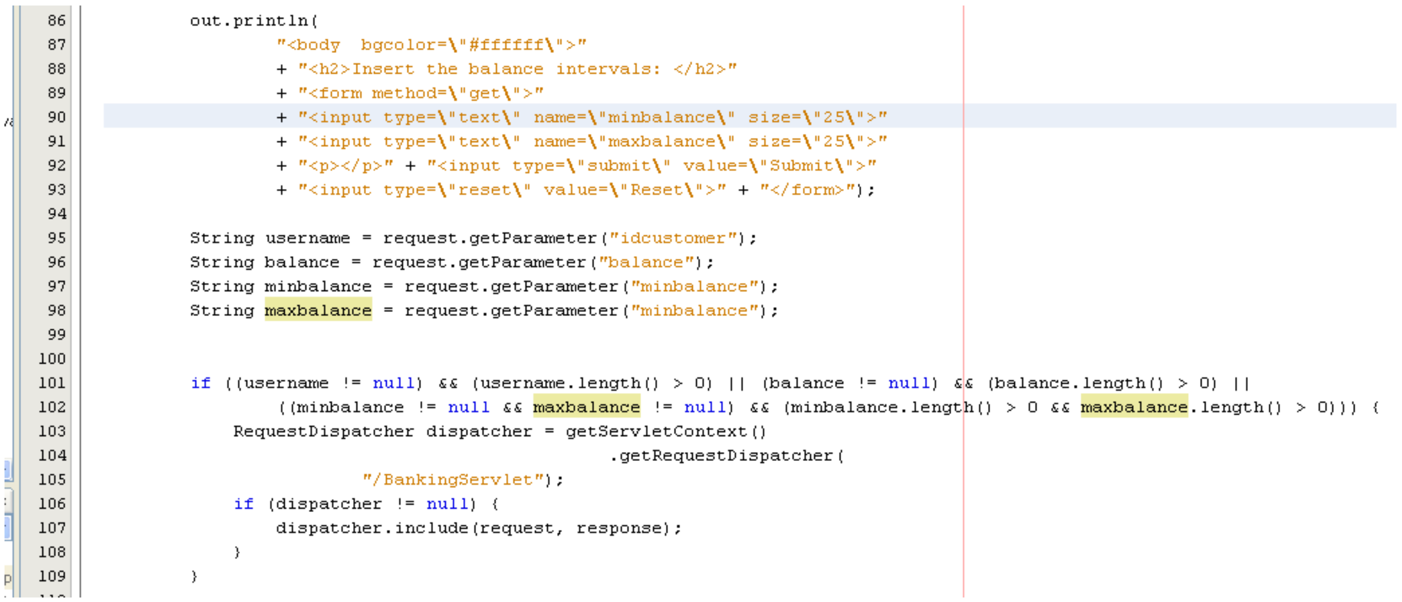
Insert the ID of the customer for which you want to know the accounts

Insert the balance:

The accounts will balance greater than: 6000 are as follows:
Account ID:1

Now we develop the interface to ask for accounts with balance that falls in a certain interval:

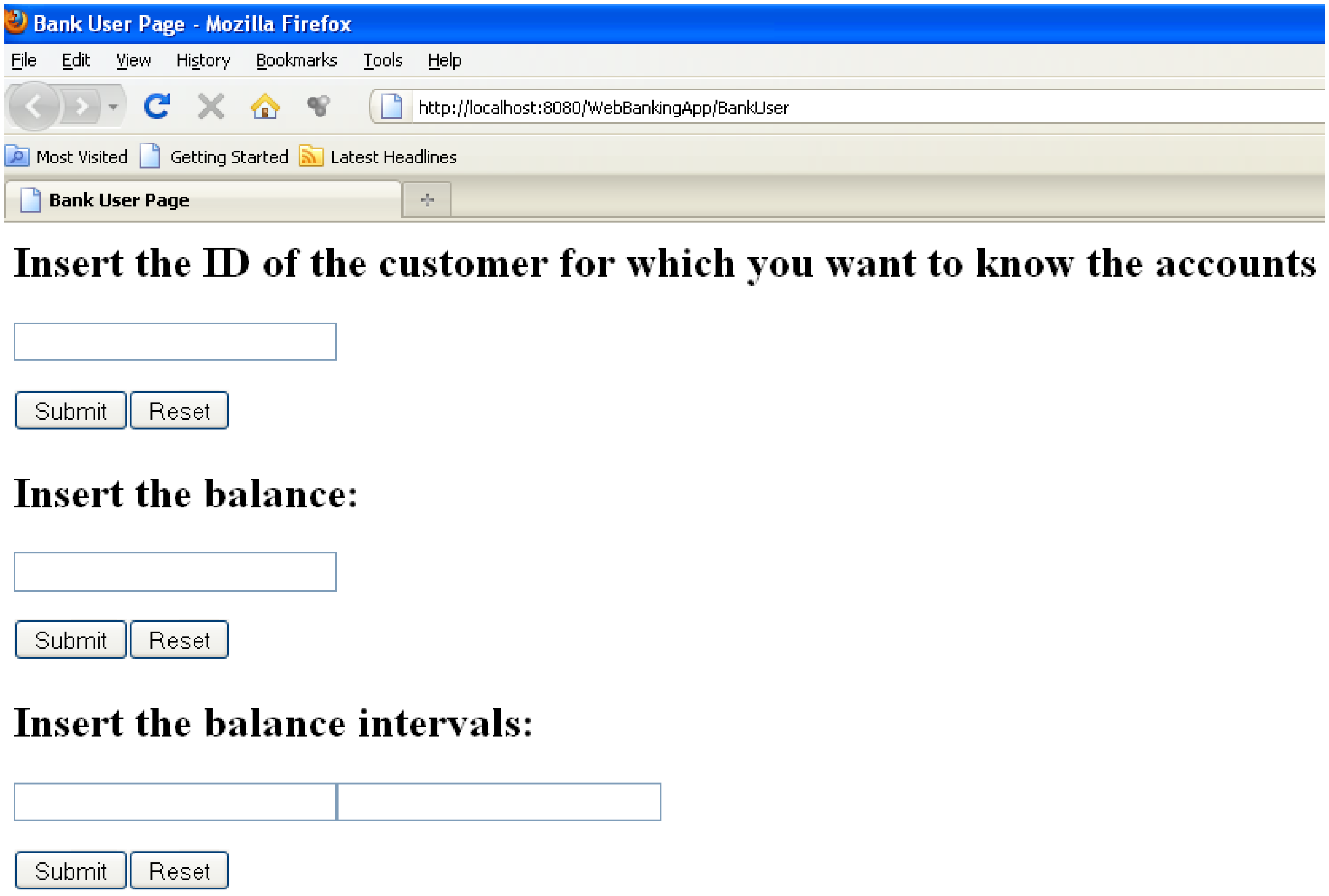
Make the following changes to BankUser:



```
        out.println(
                "<body  bgcolor=\"#ffffff\">"
                + "<h2>Insert the balance intervals: </h2>"
                + "<form method=\"get\">"
                + "<input type=\"text\" name=\"minbalance\" size=\"25\">"
                + "<input type=\"text\" name=\"maxbalance\" size=\"25\">"
                + "<p></p>" + "<input type=\"submit\" value=\"Submit\">"
                + "<input type=\"reset\" value=\"Reset\">" + "</form>");

        String username = request.getParameter("idcustomer");
        String balance = request.getParameter("balance");
        String minbalance = request.getParameter("minbalance");
        String maxbalance = request.getParameter("minbalance");


        if ((username != null) && (username.length() > 0) || (balance != null) && (balance.length() > 0) ||
                ((minbalance != null && maxbalance != null) && (minbalance.length() > 0 && maxbalance.length() > 0))) {
            RequestDispatcher dispatcher = getServletContext()
                                           .getRequestDispatcher(
                        "/BankingServlet");
            if (dispatcher != null) {
                dispatcher.include(request, response);
            }
        }
```

Note the changes in lines: 86-93, 97-98, 101-102.

If you run the file BankUser, you will get the following:


Bank User Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/WebBankingApp/BankUser
Most Visited Getting Started Latest Headlines
Bank User Page

Insert the ID of the customer for which you want to know the accounts

Insert the balance:

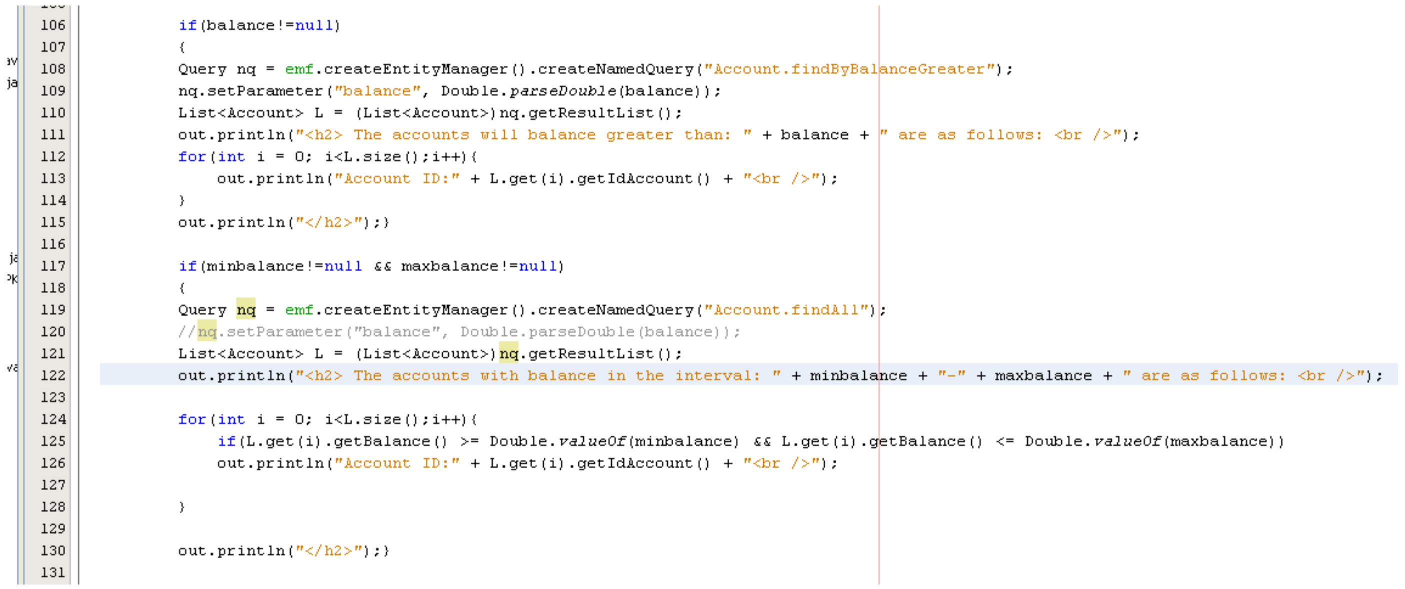
Insert the balance intervals:

Now we need to change BankingServlet:

Add the following as shown in lines 91-92:

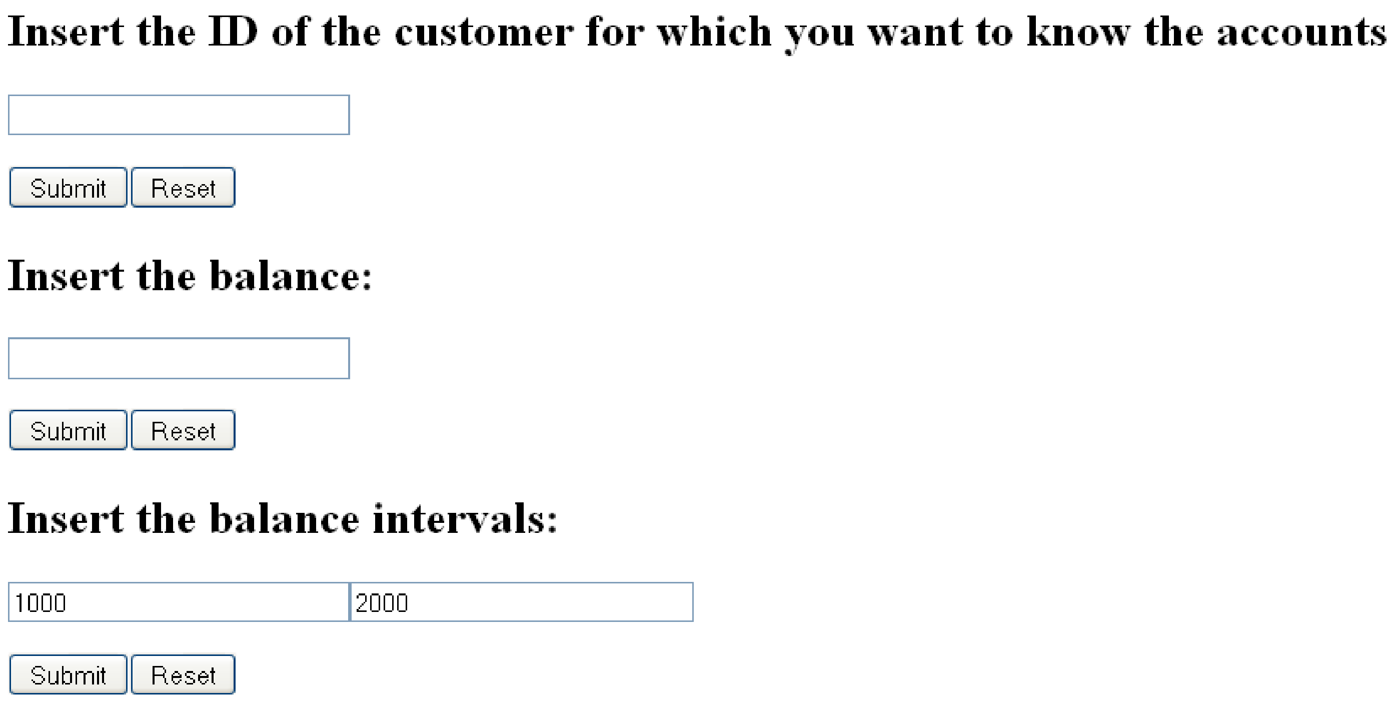
```
87              PrintWriter out = response.getWriter();
88              // then write the data of the response
89              String idCustomerFromUser = request.getParameter("idcustomer");
90              String balance = request.getParameter("balance");
91              String minbalance = request.getParameter("minbalance");
92              String maxbalance = request.getParameter("maxbalance");
93
```

Add the following code as shown in lines 119-130:



```
if(balance!=null)
{
Query nq = emf.createEntityManager().createNamedQuery("Account.findByBalanceGreater");
nq.setParameter("balance", Double.parseDouble(balance));
List<Account> L = (List<Account>)nq.getResultList();
out.println("<h2> The accounts will balance greater than: " + balance + " are as follows: <br />");
for(int i = 0; i<L.size();i++){
    out.println("Account ID:" + L.get(i).getIdAccount() + "<br />");
}
out.println("</h2>");}

if(minbalance!=null && maxbalance!=null)
{
Query nq = emf.createEntityManager().createNamedQuery("Account.findAll");
//nq.setParameter("balance", Double.parseDouble(balance));
List<Account> L = (List<Account>)nq.getResultList();
out.println("<h2> The accounts with balance in the interval: " + minbalance + "-" + maxbalance + " are as follows: <br />");

for(int i = 0; i<L.size();i++){
    if(L.get(i).getBalance() >= Double.valueOf(minbalance) && L.get(i).getBalance() <= Double.valueOf(maxbalance))
    out.println("Account ID:" + L.get(i).getIdAccount() + "<br />");

}

out.println("</h2>");}
```

Now you can run the BankUser file:



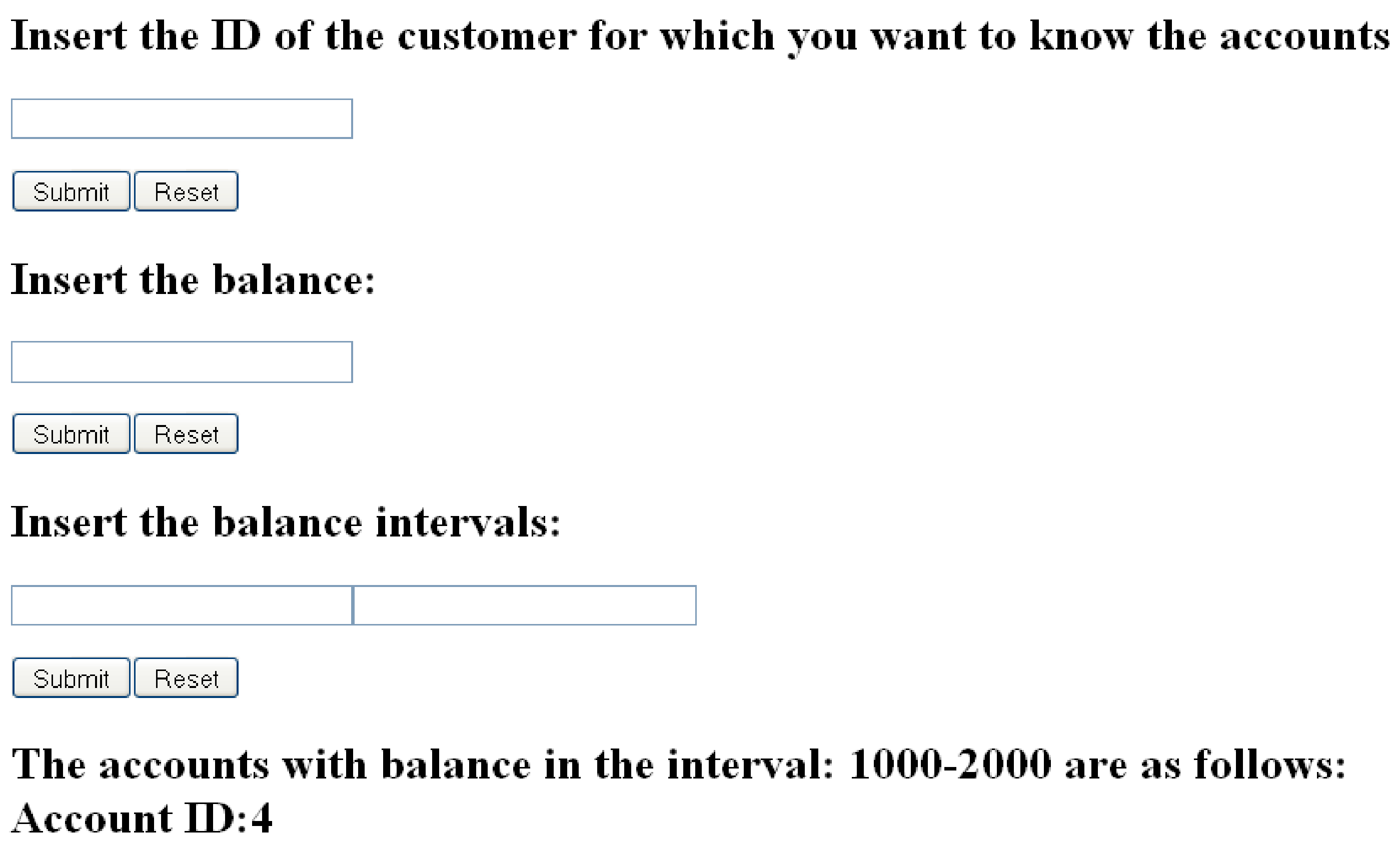
Insert the ID of the customer for which you want to know the accounts

Insert the balance:

Insert the balance intervals:

1000 | 2000

You will see the following:



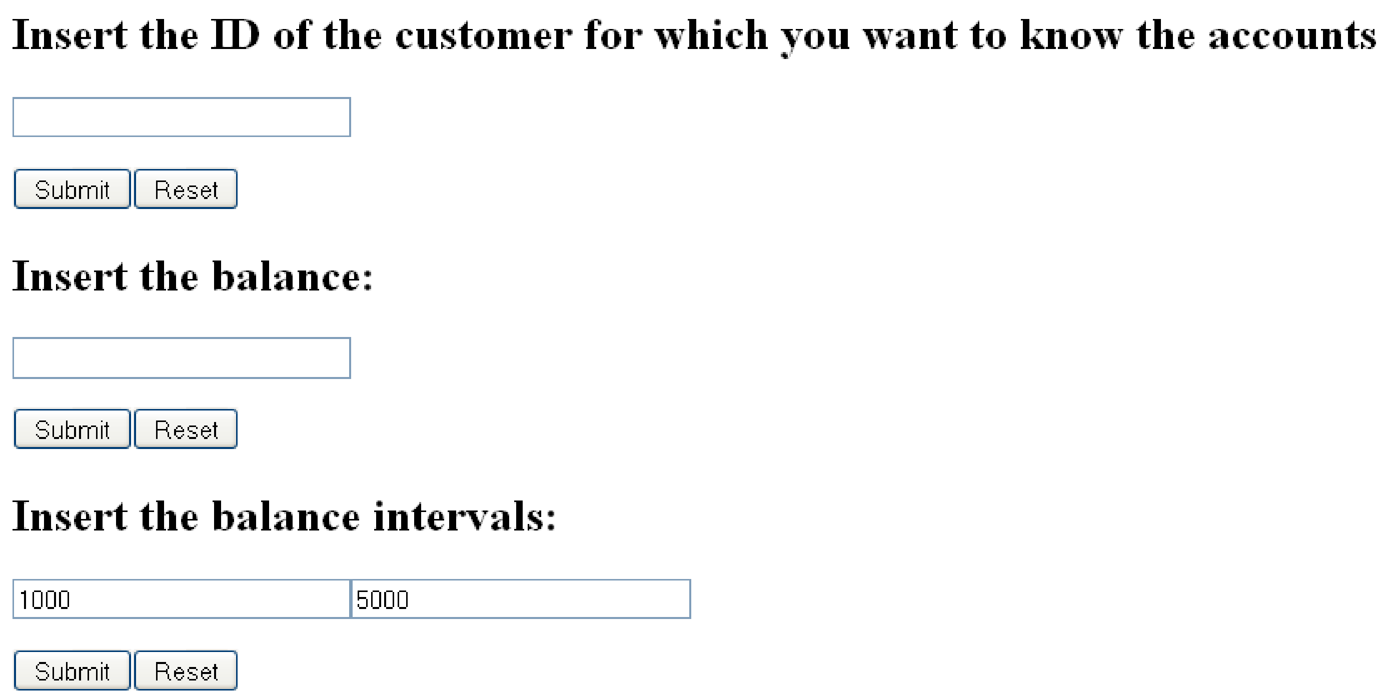
Insert the ID of the customer for which you want to know the accounts

Insert the balance:

Insert the balance intervals:

The accounts with balance in the interval: 1000-2000 are as follows:
Account ID:4

Now let us try with new values:

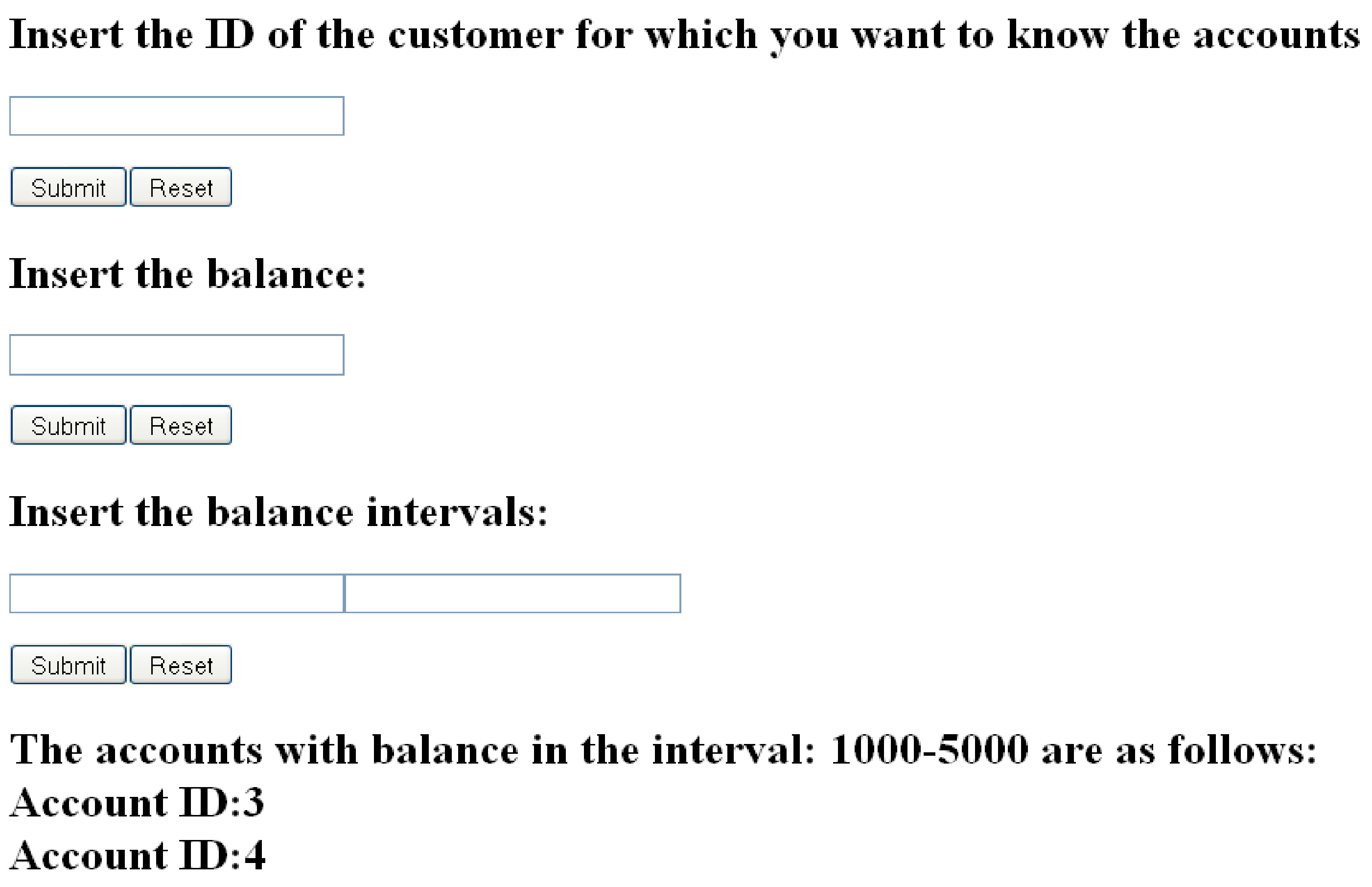


Insert the ID of the customer for which you want to know the accounts

Insert the balance:

Insert the balance intervals:

1000 5000

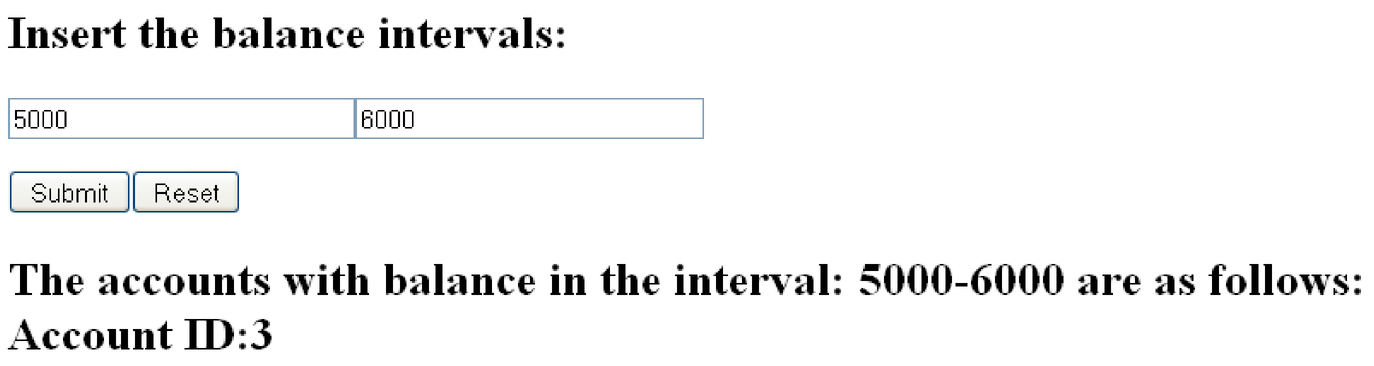


Insert the ID of the customer for which you want to know the accounts

Insert the balance:

Insert the balance intervals:

The accounts with balance in the interval: 1000-5000 are as follows:
Account ID:3
Account ID:4



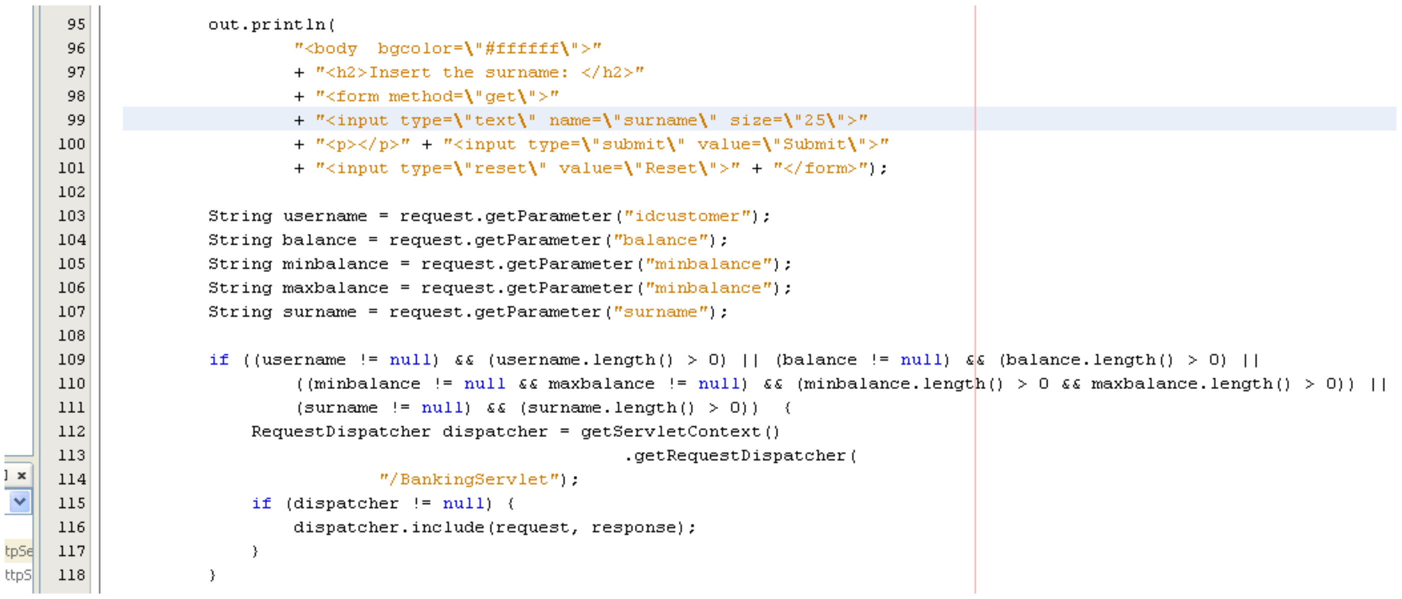
Insert the balance intervals:

5000 6000

The accounts with balance in the interval: 5000-6000 are as follows:
Account ID:3

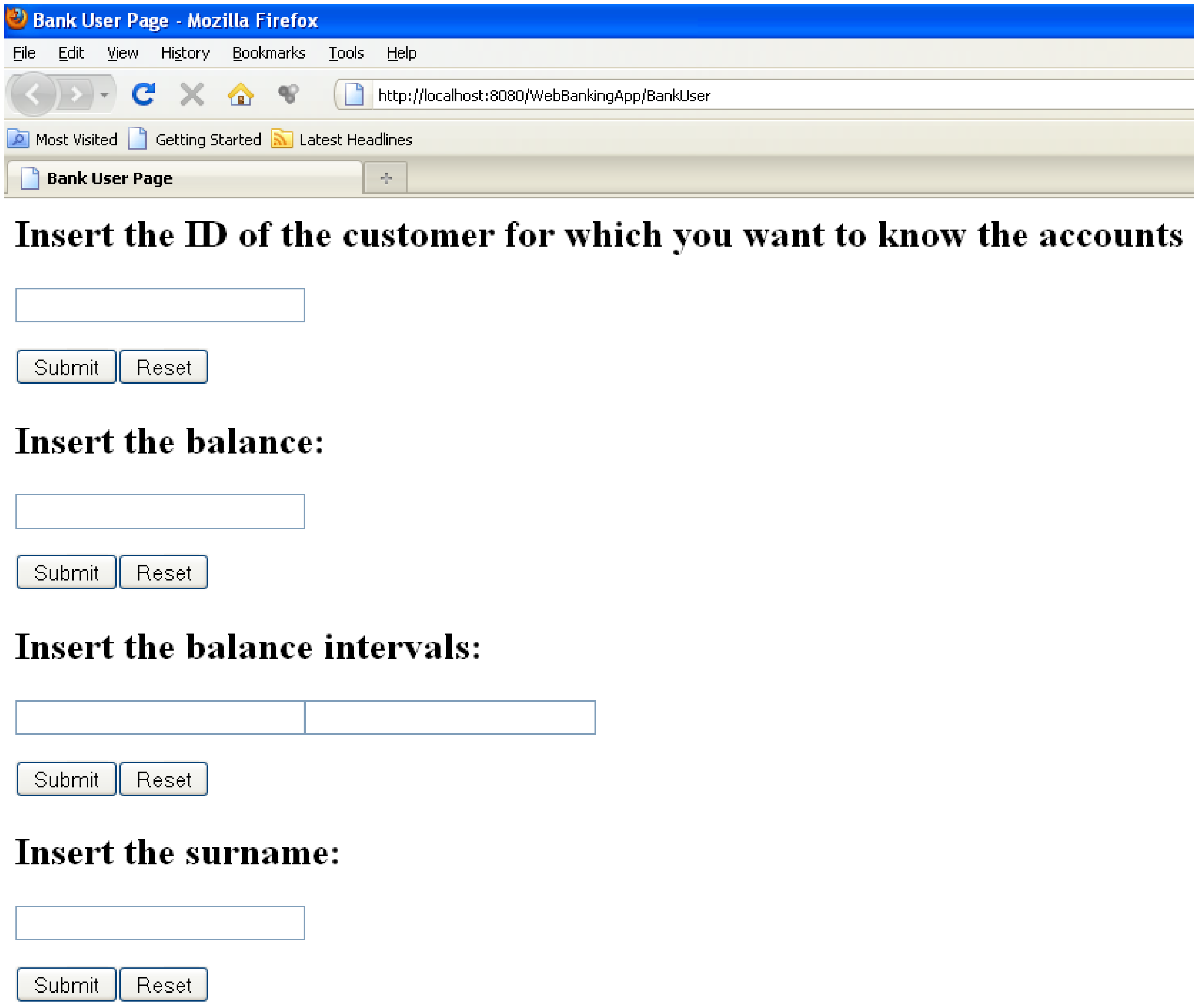
Now we develop a method that finds an account by surname of the customer:

Change the code as follows:



```
 95         out.println(
 96                 "<body  bgcolor=\"#ffffff\">"
 97                 + "<h2>Insert the surname: </h2>"
 98                 + "<form method=\"get\">"
 99                 + "<input type=\"text\" name=\"surname\" size=\"25\">"
100                 + "<p></p>" + "<input type=\"submit\" value=\"Submit\">"
101                 + "<input type=\"reset\" value=\"Reset\">" + "</form>");
102 
103         String username = request.getParameter("idcustomer");
104         String balance = request.getParameter("balance");
105         String minbalance = request.getParameter("minbalance");
106         String maxbalance = request.getParameter("minbalance");
107         String surname = request.getParameter("surname");
108 
109         if ((username != null) && (username.length() > 0) || (balance != null) && (balance.length() > 0) ||
110                 ((minbalance != null && maxbalance != null) && (minbalance.length() > 0 && maxbalance.length() > 0)) ||
111                 (surname != null) && (surname.length() > 0))  {
112             RequestDispatcher dispatcher = getServletContext()
113                                                     .getRequestDispatcher(
114                         "/BankingServlet");
115             if (dispatcher != null) {
116                 dispatcher.include(request, response);
117             }
118         }
```

Note the changes in lines 95-101, 107, and 109-111.


Bank User Page - Mozilla Firefox

http://localhost:8080/WebBankingApp/BankUser

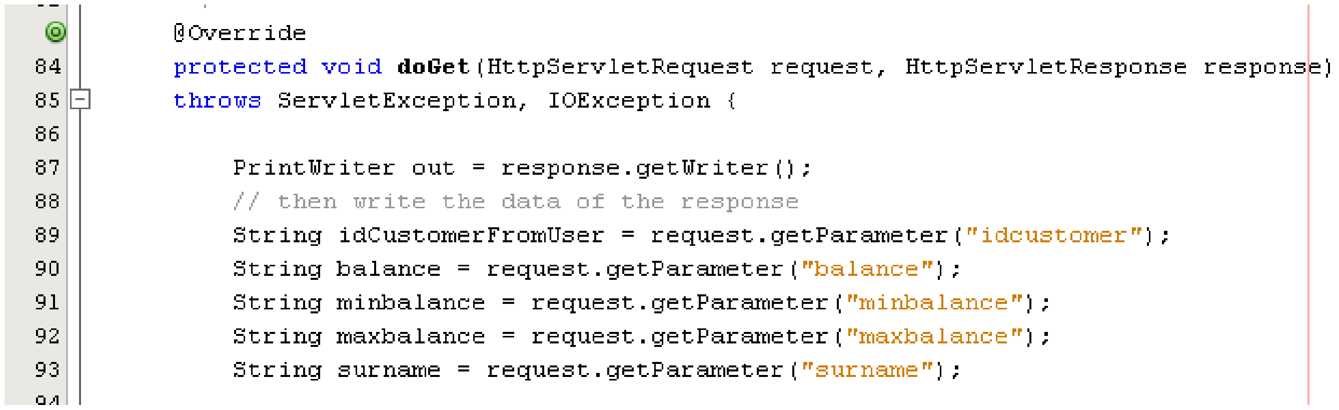
Insert the ID of the customer for which you want to know the accounts

Insert the balance:

Insert the balance intervals:

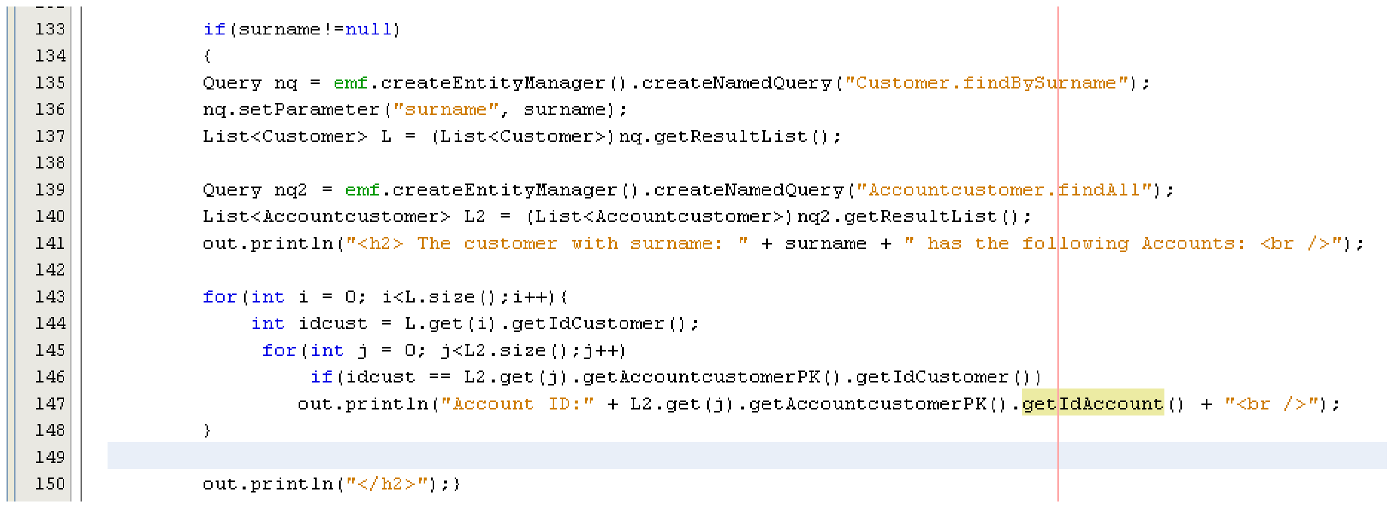
Insert the surname:

Now we change the BankingServlet class as follows:



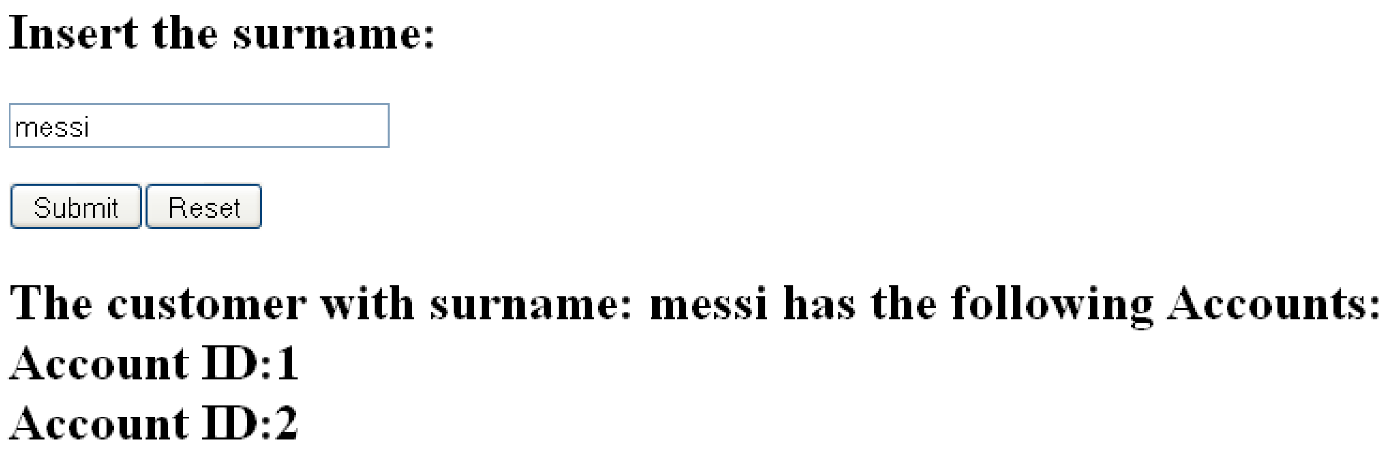
```
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

        PrintWriter out = response.getWriter();
        // then write the data of the response
        String idCustomerFromUser = request.getParameter("idcustomer");
        String balance = request.getParameter("balance");
        String minbalance = request.getParameter("minbalance");
        String maxbalance = request.getParameter("maxbalance");
        String surname = request.getParameter("surname");
```

Note the change in line 93.



```
        if(surname!=null)
        {
        Query nq = emf.createEntityManager().createNamedQuery("Customer.findBySurname");
        nq.setParameter("surname", surname);
        List<Customer> L = (List<Customer>)nq.getResultList();

        Query nq2 = emf.createEntityManager().createNamedQuery("Accountcustomer.findAll");
        List<Accountcustomer> L2 = (List<Accountcustomer>)nq2.getResultList();
        out.println("<h2> The customer with surname: " + surname + " has the following Accounts: <br />");

        for(int i = 0; i<L.size();i++){
            int idcust = L.get(i).getIdCustomer();
             for(int j = 0; j<L2.size();j++)
                 if(idcust == L2.get(j).getAccountcustomerPK().getIdCustomer())
                out.println("Account ID:" + L2.get(j).getAccountcustomerPK().getIdAccount() + "<br />");
        }

        out.println("</h2>");}
```

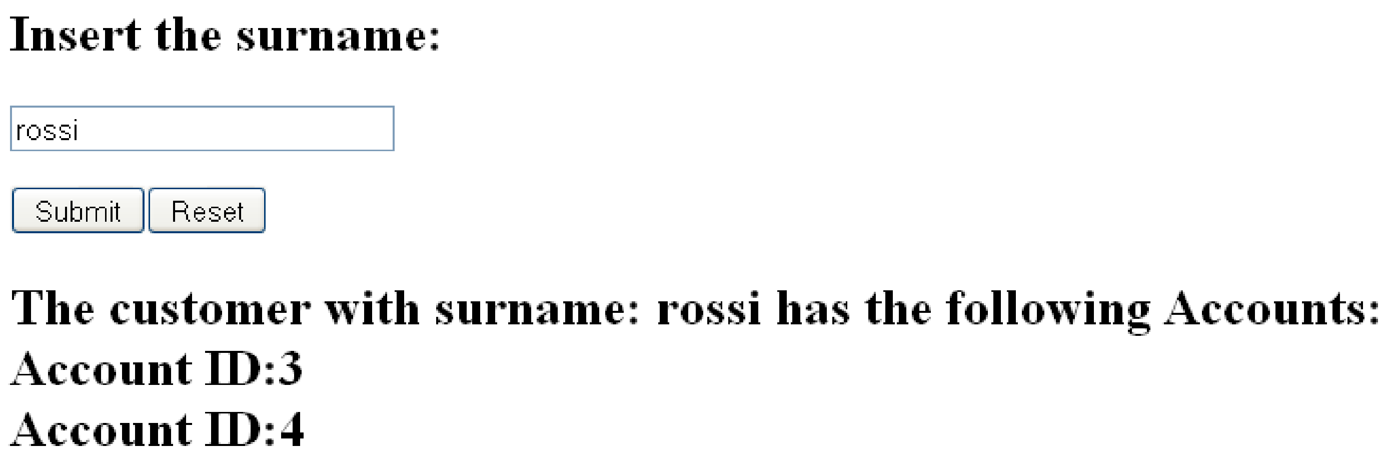
If you run the file BankUser you will have:



Insert the surname:

messi

The customer with surname: messi has the following Accounts:
Account ID:1
Account ID:2

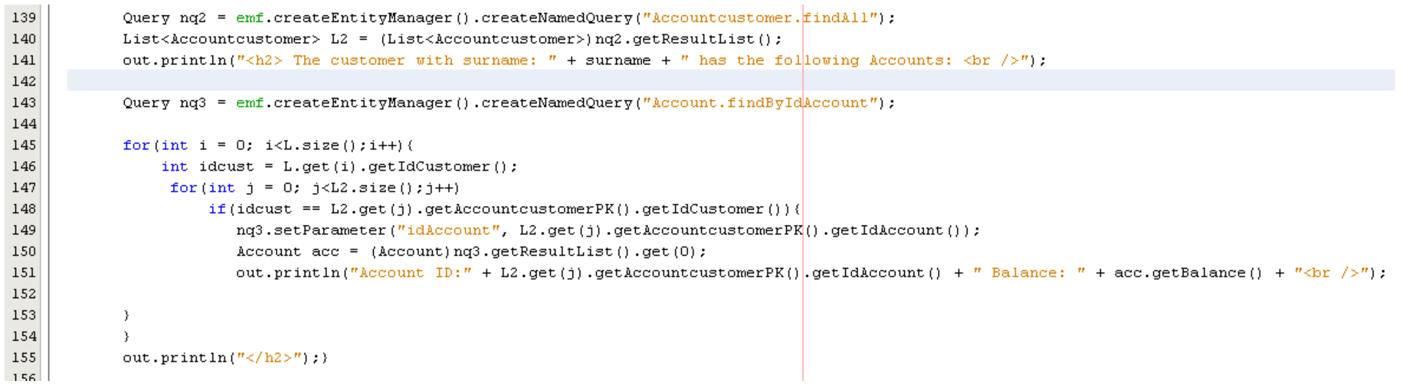


Insert the surname:

```
rossi
```

The customer with surname: rossi has the following Accounts:
Account ID:3
Account ID:4

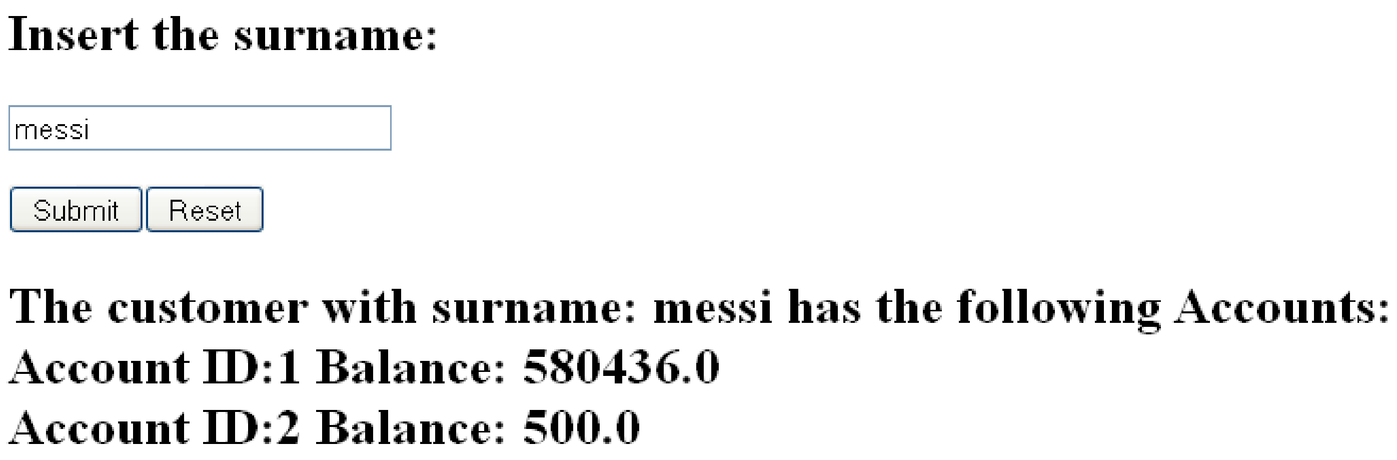
If we want to show also the balances of the accounts we need to change to code in BankingServlet as follows:



```
139         Query nq2 = emf.createEntityManager().createNamedQuery("Accountcustomer.findAll");
140         List<Accountcustomer> L2 = (List<Accountcustomer>)nq2.getResultList();
141         out.println("<h2> The customer with surname: " + surname + " has the following Accounts: <br />");
142 
143         Query nq3 = emf.createEntityManager().createNamedQuery("Account.findByIdAccount");
144 
145         for(int i = 0; i<L.size();i++){
146             int idcust = L.get(i).getIdCustomer();
147              for(int j = 0; j<L2.size();j++)
148                  if(idcust == L2.get(j).getAccountcustomerPK().getIdCustomer()){
149                     nq3.setParameter("idAccount", L2.get(j).getAccountcustomerPK().getIdAccount());
150                     Account acc = (Account)nq3.getResultList().get(0);
151                     out.println("Account ID:" + L2.get(j).getAccountcustomerPK().getIdAccount() + " Balance: " + acc.getBalance() + "<br />");
152 
153         }
154         }
155         out.println("</h2>");}
156
```

Note the changes in lines: 143, 147-154.

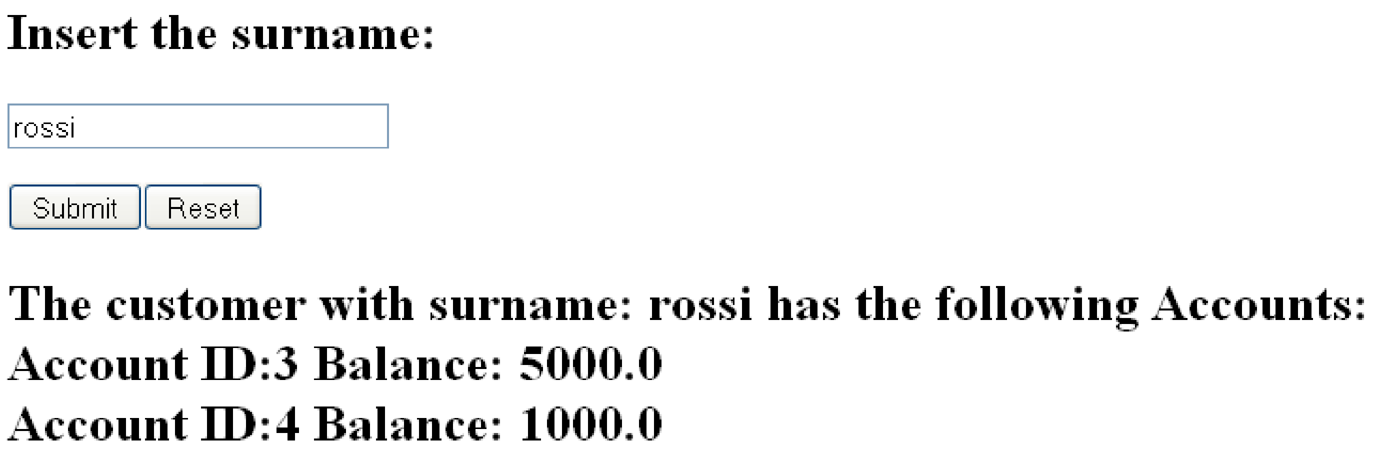
If you run the file you will get:



Insert the surname:

```
messi
```

The customer with surname: messi has the following Accounts:
Account ID:1 Balance: 580436.0
Account ID:2 Balance: 500.0



Insert the surname:

rossi

The customer with surname: rossi has the following Accounts:
Account ID:3 Balance: 5000.0
Account ID:4 Balance: 1000.0