北星学園大学経済学部北星論集第53巻第１号（通巻第64号）（2013年９月）・抜刷

A Java Application Development Platform with a Unified Modeling Language (UML) Plug-in (Part Ⅱ)

能　登　　宏

A Java Application Development Platform with a Unified Modeling Language (UML) Plug-in (Part Ⅱ)

Hiroshi NOTO

Contents

4. Case Studies of Running the Platform

4.4 NetBeans + UML + Servlet(JSP) + GlassFish + EJB

Course【Application Ⅱ [2013]】
Project: "HelloWorldProjectEJB"

In this project we display in the browser the "Hello World, Everyone!" message created in the session bean with servlets (JSP).

The purpose of the project is to grasp the basic idea of Enterprise Java Beans 3.0 (EJB 3)[15] which is one of the key Java EE specifications, providing a standard way to implement server-side components that encapsulate the business logic of our projects, in other words the (enterprise) applications.

Session Beans are one type of Enterprise Beans which are of two types, Session Beans and Massage-Driven Beans. Enterprise Beans are Java EE components that implement Enterprise JavaBeans (EJB) technology. Enterprise Beans run in the EJB container, a runtime environment within the server (e.g. the GlassFish server). The EJB container is

Key words: Java Application, Database Management System, NetBeans IDE, Unified Modeling Language (UML), Enterprise Java Beans (EJB)

nothing but a program that runs on the server and implements the EJB specifications. The EJB container provides special type of the environment suitable for running the Enterprise Beans that are used mostly in distributed applications and typically contain the business logic. The present "HelloWorldProjectEJB" project exemplifies a session bean. A session bean represents a single client accessing the (enterprise) application deployed on the application server by invoking its method.

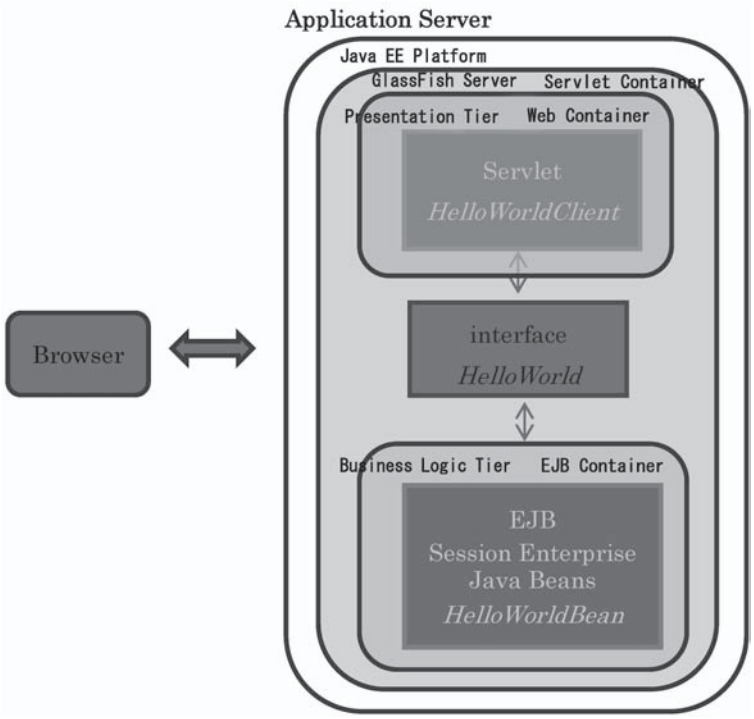

Fig. 4-22. Web browser making the client servlet display the session bean (EJB)

Although the project exhibits a simple action, we can understand the basic role of the Enterprise Java Beans (EJB) and look at the servlet designed to display the session enterprise bean. The EJB known as session enterprise beans, as explained in the previous paragraph, is a reusable, server-side piece of code which implements business logic to be used within enterprise applications and runs in the EJB container. In the web tier or the presentation tier, servlet, JSP and JSF technologies are utilized in developing the Java EE applications. In the Java EE applications the web content is served by a program running in a container with deployed servlets, JSP, JSF or by a third-party framework. A servlet is a Java program written by certain rules (HTML, Java Script, applet or the like) and deployed in a Java EE-compliant servlet container such as the GlassFish server in our case. Thus the present project has two tiers, i.e. the business logic tier and the presentation tier. A Java EE application typically employs the multitier architecture. One of the greatest advantages of the multitier architecture is its scalability. This means that we can change one component in a tier without changing the other components in the rest of the tiers.

Now let us take a look at how our simple EJB application development goes. The following subsections explain how to build, deploy and execute "Hello World, Everyone ! " session enterprise bean.

Creating Session Enterprise Beans
EJB name: "HelloWorldBean"
Package Name: helloworld.ejb
Session Type: stateless

The Enterprise Java Beans 3.0 (EJB 3) is, as explained before, one of the basic Java EE specifications, providing a standard way to implement server-side components that encapsulate the business logic of enterprise applications, executing business actions inside the EJB container of the application server. In this process we build and deploy "HelloWorldBean" session enterprise bean. A session bean can be either stateless or stateful. The stateless session beans provide business methods (namely business logic) to their clients without maintaining a conversational state with them, thereby making it possible for the EJB container to pool and reuse instances of the stateless beans, sharing them between the clients. The "outputHelloWorld" method here takes no parameters and compiles a "Hello World, Everyone!" message, returning the compiled message to the client.

List. 4-2. Simple Example of a Stateless Session Bean Class of "HelloWorldBean" Session Enterprise Bean

```
package helloworld.ejb;

import javax.ejb.Stateless;

/**
 *
 * @author NOTOHirosi
 */
@Stateless
public class HelloWorldBean implements helloworld.ejb.HelloWorld {

    // Add business logic below. (Right-click in editor and choose
    // "Insert Code > Add Business Method")
    public String outputHelloWorld() {
        return "Hello World, Everyone!";
    }
}
```

In List 4-2, the annotation @Stateless declares that the "HelloWorldBean" class is stateless. Sun Microsystem added the features like annotation to make the development easier and more efficient in jdk 5 and 6. More specifically the annotation associates the program elements with the metadata so that the compiler can extract program behavior to support the annotated elements to generate interdependent code when necessary. The *at* sign character (@) indicates to the compiler that what follows is an annotation.

Creating Java Interface
Class Name: “HelloWorld”
Package Name: helloworld.ejb

As for the session beans, it is very desirable for them to be accessed only through the business interface they define. This programming model enables us to easily update, modify, or change the bean classes without alter the client code leaving their business interfaces untouched. One more thing to be noticed is the @Local annotation with which we declare the “HelloWorld” interface as local (see List. 4−3). This means that the “HelloWorldBean” session beans and the “HelloWorld” interface are tightly coupled. In other words the session bean and the client application (see “HelloWorldClient” below) consuming that bean through the interface are running in the same JVM (Java Virtual Machine). That is why we use a local interface. Alternatively we could specify the interface in the bean’s class: when @Local is used on the beans class, it declares the local business interface for a session bean. Either will do.

List. 4−3. Interface Class of "HelloWorld"

```
package helloworld.ejb;

import javax.ejb.Local;

/**
 *
 * @author NOTOHirosi
 */
@Local
public interface HelloWorld {

    public String outputHelloWorld();
}
```

Creating Servlet
Class Name: “HelloWorldClient”
Package Name: helloworld.client

In this project we create the client application with the servlet which runs the present project on the application server. When building a web application the web container or also known as the presentation tier comes into Fig. 4−22. In this example we have our choice of servlet responsible for the look of our client “HelloWorldClient” in the presentation tier. A servlet is a Java program written by certain rules and deployed in a Java EE−compliant servlet container of our choice (i.e. GlassFish server). The client program can be a lightweight HTML page that the servlet creates as shown in List. 4−4. If the web server listens to the client’s request HttpSelvletRequest, the server processes the request and sends back HttpServletResponse with the requested static content.

List. 4-4. "HelloWorldCllient" Servlet

```
package helloworld.client;
import helloworld.ejb.HelloWorld;
import helloworld.ejb.HelloWorldBean;
import java.io.IOException;
import java.io.PrintWriter;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
/**
 * @author NOTOHirosi
 */
public class HelloWorldClient extends HttpServlet {
    @EJB
    private HelloWorld helloworld;
    /**
     * 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. You may use following sample code. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet HelloWorldClient</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet HelloWorldClient at " + request.getContextPath() + "</h1>");
            HttpSession session = request.getSession();
            out.println("<h1>Message from HelloWorldBean at " + session.getServletContext() +
"</h1>");
            out.println("<h1>Message from HelloWorldBean: " + helloworld.outputHelloWorld() +
"</h1>");
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }
```

The most interesting thing to note here in this code is the use of the @EJB annotation. We use this to annotate the static field “private HelloWorld helloworld” that declares and represents the “HelloWorld” business interface. In our case we acquire a reference to the “HelloWorld” interface by annotating the private “HelloWorld helloworld” with @EJB. This is an example of how a client can obtain a session bean’s business interface using dependency injection. Dependency injection lets the container automatically insert reference to other components and resources with the help of annotations.

We have learned in this subsection how to build a simple stateless Session Bean, deploy it to the GlassFish application server, and then execute it with a client application. The directory structure of the present project is depicted in the Projects window as shown in Fig. 4–23.

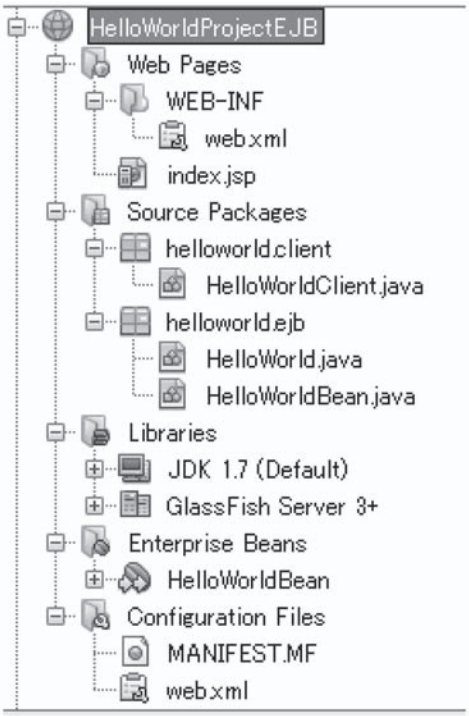

Fig. 4–23. The directory structure for the “HelloWorldProjectEJB” project.

Now that we have created and deployed “HelloWorldBean” enterprise been, we can execute it by creating a client application with the servlet that will make the EJB container create an instance of that bean and invoke its “outputHelloWorld” method through the “HelloWorld” interface. Fig. 4–24 shows the result of the servlet displaying the message from the “HelloWorldBean” together with other messages from the servlet.

Fig. 4-24. Displaying the message from HelloWorldBean by running HelloWorldClient.java

4.5 NetBeans + UML + JSF + Java DB + GlassFish + EJB

Course【Seminar Ⅱ [2013]】
Project："WebAppJUnitSolution"

In this project we test a Java EE enterprise application using the embedded EJB container. We also learn very briefly to describe Java Persistence API.[15] The Java Persistence API(JPA) represents a standard way of accessing relational databases from Java EE applications. JPA is a new Java EE technology dealing with data persistence. It brings the object-oriented and relational models together, making Java EE developers more productive. We owe this project mostly to a NetBeans tutorial.[16]

We are going to create a web application with an entity class and a session bean class. However in this project we first make a JUnit test class for the session bean and run the test in the embedded EJB container. The NetBeans IDE provides built-in support for generating and executing unit tests based on the JUnit frameworks. The JUnit framework is an open source product that supports development of tests and gives a harness for running these tests.

An embedded (or embeddable) EJB container is launched by our application when we get control, while a server EJB container is launched as part of an application server's application and usually has deeper integration with other services and technologies included with the application server. For this reason, an embeddable EJB container is convenient for unit testing like the present case.

Creating Session Beans
EJB name: "MyBean"
Package Name: "bean"
Session Type: stateless

We create a very simple session bean that contains one method that adds two numbers. In this project, however, we are going to create a test class for the session bean

that will test the “addNumbers” method. Then the IDE can generate the new test class and skeleton test methods based on the methods in the target class, i.e. “MyBean” class.

In the Projects window we see that the IDE generates the test class under the Test Packages node (as shown in Fig. 4–25). By default, the IDE generates a skeleton test method in the test class that calls javax.ejb.embeddable.EJBContainer.createEJBContainer() to create an EJB container instance. The createEJBContainer() method is one of the methods in the EJB Container class that is part of the EJB 3.1 Embeddable API. Expanding the Test Libraries node in the Projects window, we can see that the IDE automatically added GlassFish Server (embeddable container) and JUnit 4.10 as test libraries (Fig. 4–25). And if we expand the GlassFish Server library, we can see that the library contains the glassfish-embedded-static-shell.jar jar file (Fig. 4–25).

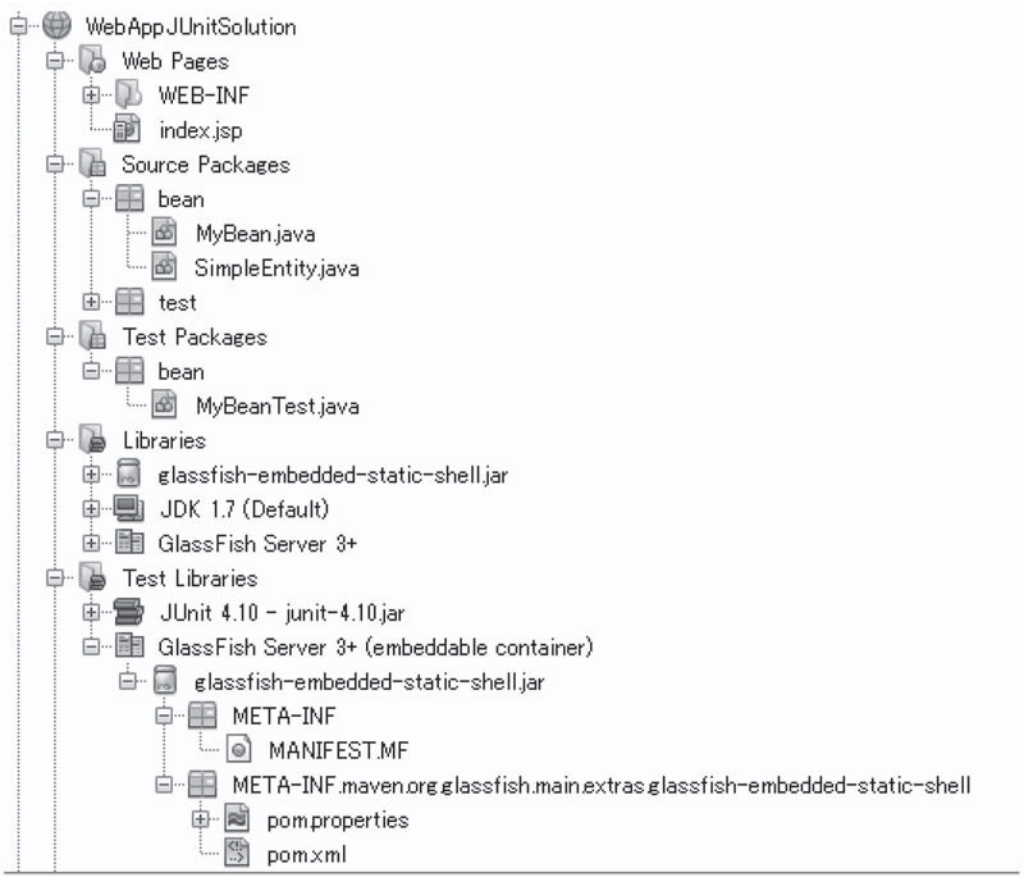

Fig. 4–25. The directory structure for the “WebAppJUnitSolution” project

Since the NetBeans IDE generated a default skeleton test class that contains code for starting the EJB container we modify the generated code that starts the container to specify additional properties for the embedded container instance. Here we display the test class in List 4–5.

List. 4-5. Test class for the session bean "MyBeanTest"

```
package bean;

import java.io.File;
import java.util.HashMap;
import java.util.Map;
import javax.ejb.embeddable.EJBContainer;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Assert;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author NOTOHirosi
 */
public class MyBeanTest {

    private static EJBContainer container;

    public MyBeanTest() {
    }

    @BeforeClass
    public static void setUpClass() throws Exception {
        Map<String, Object> properties = new HashMap<String, Object>();
        properties.put(EJBContainer.MODULES, new File("build/jar"));
        container = EJBContainer.createEJBContainer(properties);
        System.out.println("Opening the container");
    }

    @AfterClass
    public static void tearDownClass() throws Exception {
        container.close();
        System.out.println("Closing the container");
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

    /**
     * Test of addNumbers method, of class MyBean.
     */
    @Test
    public void testAddNumbers() throws Exception {
        System.out.println("addNumbers");
        int numberA = 1;
        int numberB = 2;
        MyBean instance = (MyBean)
container.getContext().lookup("java:global/classes/MyBean");
        int expResult = 3;
        int result = instance.addNumbers(numberA, numberB);
        assertEquals(expResult, result);
        container.close();
    }

    /**
     * Test of verify method, of class MyBean.
     */
    @Test
    public void testVerify() throws Exception {
        System.out.println("verify");
```

```
            MyBean instance = (MyBean)
container.getContext().lookup("java:global/classes/MyBean");
            int expResult = 0;
            int result = instance.verify();
            assertEquals(expResult, result);
            container.close();
      }

      /**
       * Test of insert method, of class MyBean.
       */
      @Test
      public void testInsert() throws Exception {
            // Lookup the EJB
            System.out.println("Looking up EJB...");
            EJBContainer container =
javax.ejb.embeddable.EJBContainer.createEJBContainer();
            MyBean instance = (MyBean)
container.getContext().lookup("java:global/classes/MyBean");

            System.out.println("Inserting entities...");
            instance.insert(5);
            int res = instance.verify();
            System.out.println("JPA call returned: " + res);
            System.out.println("Done calling EJB");

            Assert.assertTrue("Unexpected number of entities", (res == 5));
            System.out.println(".....SUCCESSFULLY finished embedded test");
      }
}
```

In this project we also create an entity class and persistence unit and modify the session bean to inject the entity manager and access the entities. We add a simple method to the new entity class that prints the id number of the entry to the output. We then add some simple methods to the session bean to create and verify entries in the database. In this course we have used Java DB database.

Creating the Entity Class

Class Name: “SimpleEntity”

Package Name: bean

We create an entity class and persistence unit with the database connection details. When developing a JPA entity, however, there is no need to include any method performing database-related operations, such as save or update. Instead we can use the EntityManager API to manipulate JPA entity instances. This prevents us from using the JDBC API directly, which means we do not need to write our own SQL code to manipulate database data. We incorporate ORM （Object Relational Mapping） annotations （@PersistentContext and @PermitAll） into an entity, applying them either to the entity’s instance variables or the properties. With ORM annotations, we describe how objects are mapped to relational tables. In this project we have a simple example of an entity in action when building our EJB JPA application. The EJB JPA application interacts with the Java DB database included in the GlassFish application server by default.

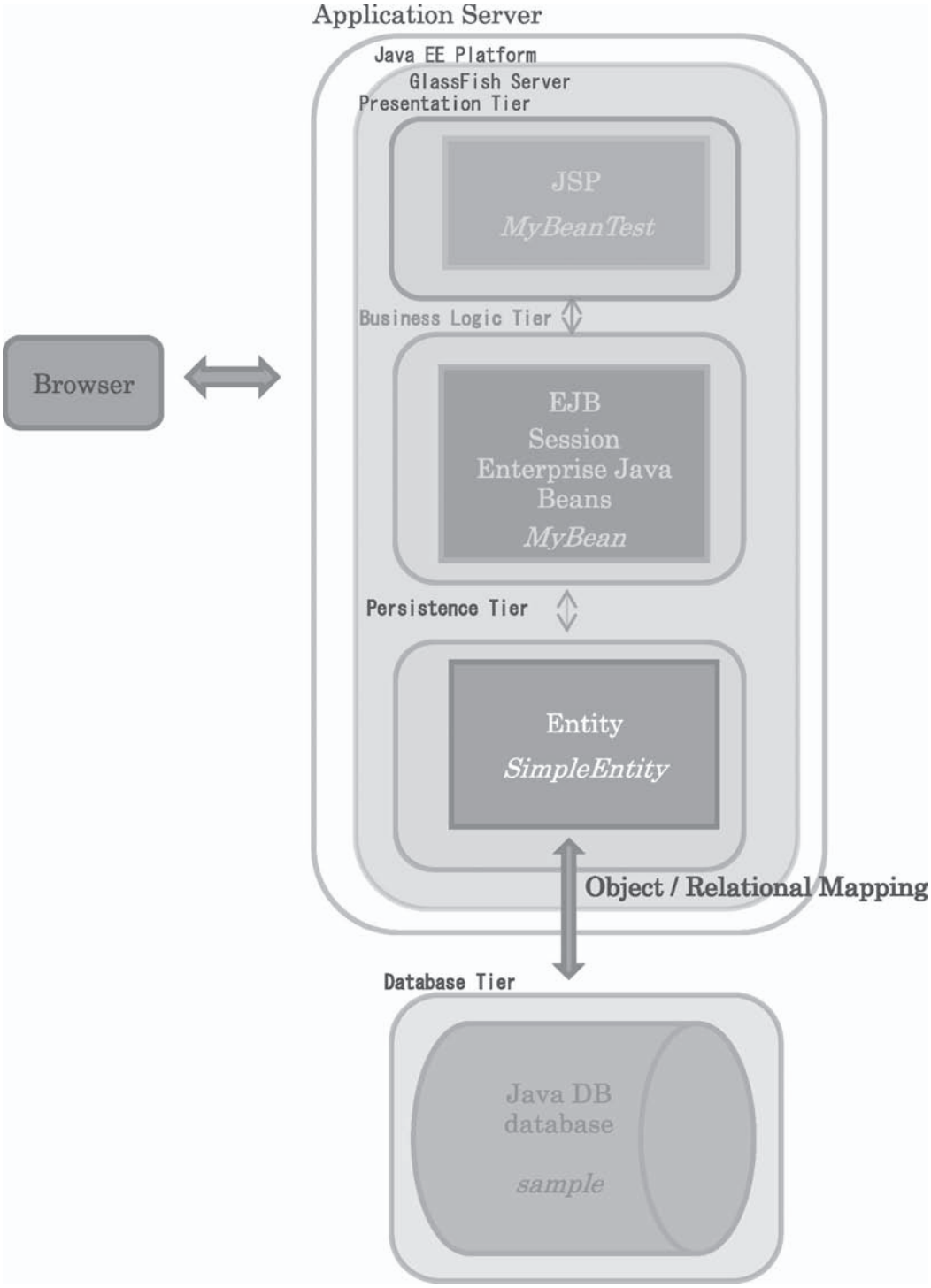

Fig. 4-26. The conceptual structure of the project "WebAppJUnitSolution" and the four-tier architecture and the Object/Relational Mapping between the entity and the database.

Now that we have started the database, we need a way to connect to it and then issue a few SQL commands against it in order to create a new database and a table within the newly created schema: we have set up the database and created the data source required to programmatically interact with that database. Then we test the JPA entity that will be mapped to the database table "sample." After modifying the "MyBean" session bean in order to add methods for inserting and retrieving data to the Java DB database table, we can test the Entity class. Finally we edit the test class to add a method to test that the application is able to look up the EJB and that the insert and verify methods are behaving correctly.

In Fig. 4–27 we show the Test Results in which the test window opens and the following messages are displayed in the TestResults window.

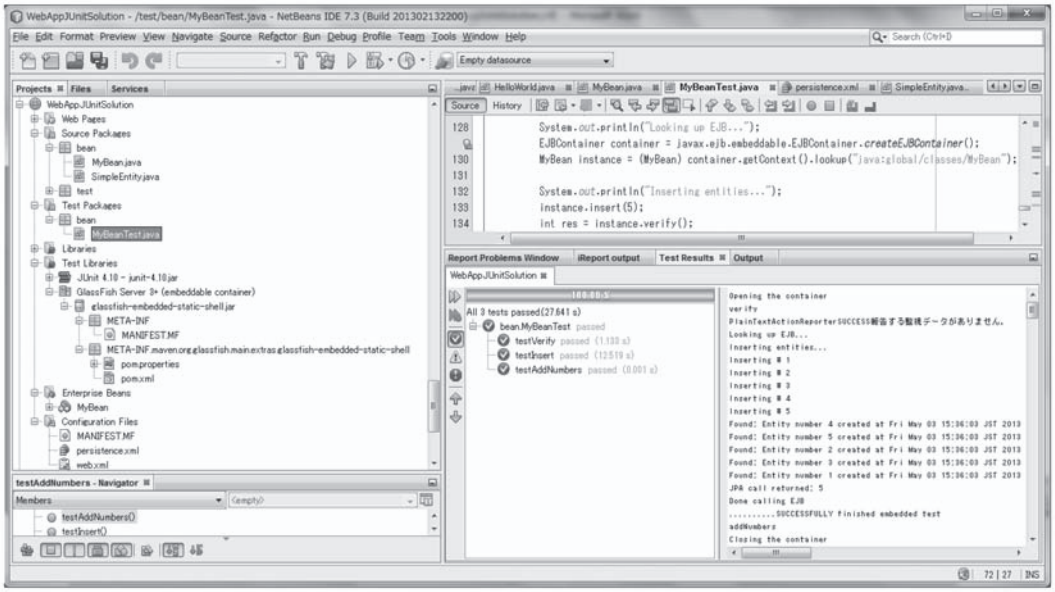

Fig. 4–27. Displaying the progress and results of the test in the Test Results window and the messages from the methods of the session bean in the Output window

5. Feedback of our Platform

Here we very briefly comment on the feedback of our Platform by the students who have taken the courses that I have been in charge of in 2012 and 2013.

In the beginning it is difficult for them to categorize or select the type of the project in the stage of domain modeling and architecture specification when they study and pursue software development.

According as the lessons and the practices are progressing, it seems that the present platform looks attractive to the students and motivates them to work on their intended architecture designs. Since all the required components or plug-ins are ready in the menu

bar that open in their respective windows, the students can set up their own software design visually and seamlessly step by step / level by level in according to each phase of their development. The present Java application development platform with UML (Unified Modeling Language) plugged in facilitates students' domain modeling appreciably, due to its ORM (Object Relational Mapping) mapping capabilities.

In the current version of NetBeans IDE based on JavaEE 6, we have emphasized the multi-tier design architecture and the connection of the IDE to a database MySQL/Java DB in our courses. Through the students' work in the software development they appeared to be gradually interested in a visual understanding of the three-tier architecture of the Model-View-Control (MVC) framework or the four-tier architecture including database tier as shown in Fig. 4-26. And finally they have become more involved in the database management process than before. Furthermore examining the four-tier diagram for the application more closely, the students could see where EJB session beans and JPA entity classes fit into the MVC structure.

The present Java platform keeps the students' motivation for and interest in software design stimulated and the students seem not to get bored with their practices on it, thereby being able to concentrate on their business logic.

The next problem, therefore, that the students are confronted with is how to code the business logic.

6. Conclusion

An Object-Oriented Java Programming Environment is introduced incorporating Unified Modeling Language (UML) as a platform on the computer system in the Computer Lab of our university.

Since all the required components or plug-ins open in their respective windows on the NetBeans platform, the processes of the system architecture of the software development are rather easy for the students to grasp and recognize visually in according to each phase of constructing their systems and the students' work flow seems not to be blocked when shifting one phase to another during their software development.

We have introduced several combinations of NetBeans and UML and their plug-ins, and elaborated the case studies of our courses which take much advantage of those configurations of the software components on the present computer platform.

In the current version of NetBeans IDE platform based on Java EE 6, the basic and core concept of the software development relies on multi-tier architecture and the modularity of the platform components which has led us to the MVC (Model-View-Controll) design framework pattern. We have also learned how to connect the IDE to a database server with the Java Persistence API (JPA) framework that allows us to manage data using object-relational mapping (ORM) in our applications built on the Java NetBeans Platform incorporating UML. Therefore in our classes we have put emphasis on the

Enterprise JavaBeans (EJB) and JPA technologies where the session beans of the application and the interactions between the tiers and modules play a very important role in the software development.

The latter half of the case studies of section 4 deals with the very matter of the EJB beans. Although the projects themselves we setup exhibit rather simple actions, we could draw a rich understanding of EJB and their containers.

The present Java platform keeps the students' motivation and interest stimulated, thereby ensuring that the students are able to concentrate on their own business logic. The problem to come that students are confronted with is how to code the business logic properly.

Acknowledgement

We thank the staffs of the Information Systems Center of Hokusei Gakuen University for constructing the computer network system with specific software and plug-ins that meets our requirement for carrying out the software development in our courses. Thanks are also due to the Visual Paradigm Company who provides the license to use the VP-UML EE 10.3 for the UML tool. We are very much obliged to the NetBeans Community for our utilizing its free and user-friendly IDE. We would also like to appreciate the Department of Management and Information of Hokusei Gakuen University for supporting us financially for getting the license to activate the software of Visual Paradigm Company.

References

1) Mary Campione and KathyWalrath. The Java Tutorial Second Edition: Object-Oriented Programming for the Internet, Sun Microsystems, Addison Wesley, 1998.
Cay S. Horstmann, Gary Cornell. Core Java Volume I and II, eighth edition, Prentice Hall, 2008.
2) Tom Pender. UML Bible, Wiley Publishing Inc., 2003.
3) Tim Howard. The Smalltalk Developer's Guide to VisualWorks, Cambridge University Press, 1998.
http://c 2.com/cgi/wiki?DomainModel: A DomainModel is an object model of a problem domain. Elements of a domain model are DomainObject classes, and the relationships between them.
4) http://netbeans.org/
5) http://www.visual-paradigm.com/
6) http://dev.mysql.com/
7) http://community.jaspersoft.com/project/ireport-designer
8) http://docs.oracle.com/javaee/
9) http://glassfish.java.net/
10) http://www.icesoft.org/java/
11) http://www.java.com/en/download/faq/develop.xml
12) Heiko Boeck. The Definitive Guide to NetBeans Plaform 7, Apress, 2012.
13) http://netbeans.org/kb/docs/web/mysql-webapp.html
14) http://www.oracle.com/technetwork/java/index-jsp-135995.html
15) The Java EE 6 Tutorial: http://docs.oracle.com/javaee/6/tutorial/doc/.
Yuli Vasiliev. Beginning Database-Driven Application Development in Java EE using GlassFish,

Apress, 2008.
16) https://netbeans.org/kb/docs/javaee/javaee-entapp-junit.html

[Abstract]

A Java Application Development Platform with a Unified Modeling Language (UML) Plug-in (Part Ⅱ)

Hiroshi NOTO

We have introduced a Java application development platform with a Unified Modeling Language tool plugged-in in the computer rooms of the Information Systems Center of Hokusei Gakuen University. The purpose of the set-up of our platform is so that the students of our courses are able to manipulate visually their own logic in the domain modeling by trial and error on this platform to decide their business models and architecture specifications which are to result in their own applications. In our seminars and lectures, we have adopted Java as the programming language and NetBeans as an integrated development environment (IDE) in Java to take advantage of NetBeans' features to help develop Java applications efficiently. The basic design of our platform complex is presented. Each configuration of several combinations of platform components is elaborated on according to each course. We present case studies of actually running the platform in our courses.

Key words: Java Application, Database Management System, NetBeans IDE, Unified Modeling Language (UML), Enterprise Java Beans (EJB)