



DEPARTMENT OF MATHEMATICS  
AND  
COMPUTER SCIENCE  
TECHNICAL REPORT SERIES

Software Components and Enterprise Javabeans

by

Gary Huband  
Department of Mathematics and Computer Science  
Georgia Southern University, Statesboro, GA 30460-8093

Number 2002-002  
Submitted: January 16, 2002  
© 2002

GEORGIA SOUTHERN UNIVERSITY

SOFTWARE  
COMPONENTS AND  
ENTERPRISE  
JAVABEANS

---

GARY HUBAND  
DECEMBER 2001

---

## TABLE OF CONTENTS

---

<b>TABLE OF CONTENTS .....</b>	<b>II</b>
<b>ABSTRACT.....</b>	<b>III</b>
<b>INTRODUCTION.....</b>	<b>1</b>
<b>SOFTWARE COMPONENTS .....</b>	<b>1</b>
DEFINITION.....	1
UNIT OF COMPOSITION .....	2
CONTRACTUAL INTERFACES .....	2
EXPLICIT CONTEXT DEPENDENCIES .....	3
THIRD PARTY COMPOSITION.....	3
<b>ENTERPRISE JAVABEANS .....</b>	<b>4</b>
DEFINITION.....	4
UNIT OF COMPOSITION .....	4
CONTRACTUAL INTERFACES .....	5
EXPLICIT CONTEXT DEPENDENCIES .....	5
THIRD PARTY COMPOSITION.....	6
EXAMPLES .....	6
<b>CONCLUSIONS.....</b>	<b>10</b>
<b>REFERENCES.....</b>	<b>11</b>
<b>APPENDIX A: HELLO APPLICATION.....</b>	<b>12</b>
HELLO BEAN .....	12
<i>Remote Interface Code</i> .....	12
<i>Home Interface Code</i> .....	12
<i>Bean Code</i> .....	12
DEPLOYMENT DESCRIPTOR.....	13
CLIENT JSP.....	13
<b>APPENDIX B: GRADES MANAGER.....</b>	<b>14</b>
GRADESMANAGER BEAN.....	14
<i>Remote Interface Code</i> .....	14
<i>Home Interface Code</i> .....	14
<i>Bean Code</i> .....	14
STUDENT BEAN .....	15
<i>Remote Interface Code</i> .....	15
<i>Home Interface Code</i> .....	16
<i>Bean Code</i> .....	16
DEPLOYMENT DESCRIPTOR.....	19
CLIENT HTML/JSP .....	20

---

## **ABSTRACT**

---

Software component development has been discussed in the literature for many years, but only recently have technologies appeared on the market. The three most prevalent today are CORBA, COM+, and Enterprise JavaBeans. Ideally, Enterprise JavaBeans allows an enterprise application assembler to purchase components, then assemble and deploy them as a fully functional application with minimal programming. There are many books on implementing Enterprise JavaBeans, but none discuss how it fits into software component development. This paper does this by first presenting definitions and characteristics of a software component. The paper then compares Enterprise JavaBeans to software component theory to see how well the implementation meets these definitions and characteristics and how it differs.



---

## INTRODUCTION

---

*“...the growing heterogeneity of hardware architectures and diversity of operating system and communication platforms make it difficult to build correct, portable, efficient, and inexpensive applications from scratch.” [5]*

This problem, which some thought would be solved by object-oriented languages, prompted the development of the next level of software construction, component technologies. Component technologies, ideally, allow an application developer to buy software pieces (components), and with a minimum of programming assemble a complete application by deploying the components into an application framework. Software development is then broken into framework developers, component developers, application assemblers, and application deployers.

Component application development is closer to the way other types of products are developed. If I sell custom computers I don't have to design and build the CPU, hard drive, motherboard, etc. from scratch. I can buy each of the components and then assemble them into a fully customized computer system.

Currently there is not one component technology for all software domains. For instance, GUI developers have Visual Basic controls and JavaBeans. In the last few years both Microsoft and Sun have developed component technologies for the enterprise (business) domain. In particular, Sun's Enterprise JavaBeans (EJB) has gained popularity primarily because of its promise to be platform independent. There are many books on implementing Enterprise JavaBeans [8][11], but none discuss how it fits into software component theory.

This paper accomplishes this by first presenting component theory through component definitions and characteristics. The paper then compares EJB to this theory to see how well it meets these definitions and characteristics and how it differs.

---

## SOFTWARE COMPONENTS

---

### DEFINITION

*“A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.” [9]*

A component should have interfaces that conform to a specification allowing components to communicate when plugged into a framework. The component's dependence on external services should also be explicitly specified. Finally, a component can be developed in-house, but ideally would be obtained from a third party in a binary form.

The next sections briefly discuss each part of this definition and other characteristics of components.

## **UNIT OF COMPOSITION**

A component is a collection of software pieces that can be independently deployed and can interact with other components to form a functioning software application [9].

A component is obtained by the component assembler in a binary form, should be completely independent of other components, and should be coarse-grained. For example, a component that provides scientific or business graphs would only need the data to graph; it would not rely on another component for any graphing functionality.

Ideally a component is a black-box system; that is the component assembler can only see the component interfaces, not the implementation. The component provides services (data or actions) to the client (another component or external software) through the component's interfaces.

## **CONTRACTUAL INTERFACES**

Currently, there are no component-oriented languages [12]. The most popular component architectures use object-oriented (OO) languages like C++ or Java. A component in these architectures is constructed from OO language objects using OO design techniques. Thus component interactions are limited to the interactions allowed by the OO language. Interactions take place through interfaces and since components are developed independently they should use interface contracts enforced by the framework. These contracts specify how the component can communicate with the framework and other components.

An interface contract is a specification that states exactly what pre-conditions the client component must establish before requesting an operation from a provider component and what post-conditions the provider must meet to supply the operation. The provider can rely on the pre-conditions being met before the operation call, and the client can rely on the post-conditions being met when the operation is completed.

The client typically initiates the action through a function or method call to the provider. The pre-conditions are the function or method call name along with the correct arguments. The provider then meets the post-condition by returning the data or returning an error or exception to the client.

There are two types of interfaces: direct and indirect. A direct interface uses a direct method call to the provider. An indirect interface makes the method call through a proxy (usually handled by the framework) then the proxy makes a direct call to the provider.

## **EXPLICIT CONTEXT DEPENDENCIES**

Context dependencies are the services a component needs from its environment. These include services of other components, services from the framework, services from the operating system, etc. Explicit means these dependencies are clearly defined.

If a component has no context dependencies it is completely self-contained and can work in any component domain. This is not practical and defeats the purpose of software reuse. Today's components have context dependencies that limits the environment where the component can exist. ActiveX controls can only run on Microsoft operating systems and JavaBeans can only be used in a Java environment.

## **THIRD PARTY COMPOSITION**

A component should be in a binary form and the component should be a black-box to the assembler. The assembler has to integrate all of the components into an application allowing the components to communicate while optimizing reuse of the components. Because current component technologies are based on Object-Oriented (OO) languages, reusing components is based on OO reuse methodologies. The commercial component model (COM+, EJB, etc.) allows these OO objects to work together as a component [12]. There are two methods of integrating and reusing components: inheritance and composition.

Inheritance, an 'is-a' relationship, allows a programmer to easily reuse another component's implementation and/ or interface. In implementation inheritance a sub-component automatically inherits state and functionality from its super-components. With interface inheritance the sub-component inherits just the interfaces from the super-components; the programmer provides the implementation. Inheritance establishes static component relationships; the relationships cannot be changed at run-time. Inheritance can greatly increase code reuse, but it can also cause problems and the utility of inheritance of components is currently being debated [2]. See Szyperski, 1998 for a full discussion of the consequences of using inheritance with components.

In composition an outer component contains a reference to an inner component. When the outer component needs a service provided by the inner component, it uses the reference to the inner component to make a method call. Composition is the most commonly used relationship between components. Composition encompasses many design methods – aggregation, delegation, and association. With aggregation, an 'is-part-of' relationship, the outer component has a reference to the inner component as an attribute. With delegation, an 'is-a-role-of' relationship [4], some of the methods of the outer component are wrapped method calls to the inner component. Delegation is a more general method of reuse and has the same problems as inheritance [9]. With association, a 'uses-a' relationship, the inner component reference only exists within an outer component method.

---

## ENTERPRISE JAVABEANS

---

### DEFINITION

*“The Enterprise JavaBeans architecture is a component architecture for the development and deployment of component-based distributed business applications.” [7]*

More specifically, Enterprise JavaBeans (EJBs) contain the business logic for a three-tier application. A component in EJB is called a bean. EJB is not a component-oriented language; currently there are no component-oriented languages [12], CORBA IDL describes interfaces only.

The component developer only writes the methods to implement the business logic and the required interfaces for the EJB container. Another party develops the container (framework). The container provides services for the component’s life cycle, transactions, persistence, multiple instance management, concurrent requests, and security [11]. As long as the container and component conform to the EJB specification [1][7], then the component will work in any EJB container. The most recent EJB specification is 2.0 [1], but the discussion below concentrates on EJB specification 1.1 since it is the most common implementation.

### UNIT OF COMPOSITION

EJBs consist of Java classes. These classes provide the interfaces to the business logic and are packaged together in a java archive (jar) file to form a component. The jar file is an independently deployable byte-code executable and can interact with other EJB components when deployed (plugged) into a container. The jar file is initially produced by the bean developer and contains one or more beans. The application assembler may then add information describing how the beans are deployed as a single application. The application deployer then deploys the application using the jar file and a container deployment tool.

The EJB specification [7] does state that EJB’s should be coarse-grained, but enforcement of this rule is left to the programmer.

Although packaged as a black-box system, a component assembler does have access to the classes in the jar file. An assembler can use the Java jar utility to see and extract all of the component classes and descriptor files. The programmer can even recover the original source code using a decompiler, change the source code, compile and add the class back into the jar file.

There are two types of EJB’s: Session and Entity\*.

---

\* EJB 2.0 includes a Message-Driven bean for asynchronous communications.

A Session EJB performs tasks that are usually independent, for instance it can coordinate the workflow for other EJBs. Session EJB's are either stateless or stateful. Stateless Session EJB's have no data members to preserve state between method calls therefore the methods are independent and the EJB can be shared among clients. Stateful session EJB's hold client data between method calls; the EJB methods may be interdependent therefore are dedicated to one client for the life of the bean [8].

Entity EJB's represent data in a database; changes to the data must be synchronized with the database (persistence). An entity EJB's persistence can be automatically managed by the container (container-managed) or the EJB can explicitly manage the persistence (bean-managed).

## **CONTRACTUAL INTERFACES**

An EJB component must interface with software outside the container, components within the container, and the container. The EJB container treats software outside the container and components within the container as clients which request a service from a component through the container and receive the service from the container. The request is in the form of a Java method call. The client must have a reference to the component, know the method name, the arguments the method expects, and the type of data returned. The container then calls the method of the component and returns the data or an exception. The arguments and return types must be legal types for Java RMI/IIOP [7] – primitive types or objects of classes that extend the Java Remote class.

Each EJB component has a contractual interface with the container. To communicate with the container each EJB has a set of interfaces it must define and implement - the home interface and the remote interface.

The home interface specifies the EJB's life-cycle method interfaces to find, create, and remove the bean. The component home class must extend the `javax.ejb.EJBHome` class. The container implements the methods of the home class.

The remote interface specifies the business method interfaces to clients. The component remote interface class must extend `javax.ejb.EJBObject`.

The component also has a bean class that defines the functionality of the business methods. The bean class must implement the appropriate bean interface, `javax.ejb.SessionBean` for session beans or `javax.ejb.EntityBean` for entity beans. For the session bean this means the component developer must include in addition to the business methods the following life-cycle methods: `ejbCreate()`, `ejbRemove()`, `ejbActivate()`, `ejbPassivate()`, and `setSessionContext()`.

## **EXPLICIT CONTEXT DEPENDENCIES**

Again, because EJB is based on Java it has the same limitations on context dependencies. It explicitly reveals the services it provides, but not the services it needs from other components. The Java reflection API provides classes that can determine the fields and methods available from a class. But, there are restrictions since an EJB cannot access

information on a class in a way that violates the Java security rules [7]. While an EJB component mostly depends on the container for services, the component can bypass the container and obtain the service itself. For example, entity beans with bean-managed persistence do not rely on the container for database synchronization.

### **THIRD PARTY COMPOSITION**

The EJB specification does not enforce a strict way of integrating components; this is left for the developer [2]. This increases the flexibility of the EJB component model but also decreases the chance of a component integrating properly with other components. The EJB specification only guarantees that a bean that follows the specification will work in any compliant container, not that the bean will work with another bean.

The EJB specification does not include inheritance as a form of connecting beans, but it may be included in a future version [2]. Some container vendors have included support for bean inheritance, such as BEA's WebLogic [13]. The following code segments shows how a child session bean would inherit from a parent session bean:

```
public class ChildBean extends examples.ejb.subclass.parent.ParentBean
    implements SessionBean {...}
```

```
public interface Child extends examples.ejb.subclass.parent.Parent {...}
```

So, the component bean class inherits functionality from the parent's bean class and inherits the interface from the parent's remote interface class.

Composition is the most common method for connecting components. To use composition a class of an EJB bean must obtain a reference to the inner component using the Java Naming and Directory Interface (JNDI). JNDI allows a client to obtain a reference to an EJB bean home interface using different naming and directory services such as LDAP, Novell Netware NDS, and CORBA Naming Service [8]. The outer bean then uses the home interface's `create()` method to obtain a reference to the inner bean. With this reference the outer bean can call any of the inner bean's public business methods.

### **EXAMPLES**

Enough of theory and specifications, let's look at an example EJB application. In this example we'll explore how to write simple EJB beans, how to connect the beans, how to deploy the beans in a container, and how to write a client to use the application.

First let's develop a very simple EJB application to display a message in a web browser. In this example we'll develop and deploy a very simple stateless session bean and develop a client web page using Java Server Pages. See Appendix A for complete code listings and deployment and compilation directions.

The bean has only one business method `sayHello()` that returns the string "Hello There!". For this stateless session bean we only have to write code for the remote interface, home interface, and bean implementation. Listing 1 is the code for the remote interface.

```
//Hello.java
package edu.gasou.cs.hellobeans;
import javax.ejb.*;
public interface Hello extends EJBObject
{
    String sayHello() throws java.rmi.RemoteException;
}
```

*Listing 1: Hello Bean Remote Interface*

The remote interface class name is the bean name, in this case Hello. The package line uniquely identifies this bean since another developer could use the same bean name. The bean has one business method `sayHello()` which takes no arguments and returns a string. All business methods must throw `RemoteException`.

Listing 2 is the code for the home interface. The class name is usually the bean name with Home attached. The container only needs the `create()` method to manage a stateless session bean. A stateless session bean must have only one `create()` method with no arguments. This method returns a reference object to the bean and must throw a `RemoteException` and a `CreateException`.

```
//HelloHome.java
package edu.gasou.cs.hellobeans;
import javax.ejb.*;
public interface HelloHome extends EJBHome
{
    public Hello create() throws java.rmi.RemoteException,
    javax.ejb.CreateException;
}
```

*Listing 2: Hello Bean Home Interface*

Next, the developer writes the bean class to implement the business methods as shown in Listing 3. The class name is usually the bean name appended with Bean. The class must have a `SessionContext` object and `setSessionContext()` method so the bean can obtain information from the container such as the home and remote interfaces, identity of the bean invoker, and transaction information. No code is needed for the life cycle methods of this session bean, but they still must be implemented. Finally, the developer writes a business method for each method defined in the remote interface.

```

//HelloBean.java
package edu.gasou.cs.hellobeans;
import javax.ejb.*;
import javax.naming.*;
public class HelloBean implements SessionBean
{
    SessionContext ctx;
    public void ejbCreate() {}
    public void ejbRemove() {}
    public void ejbPassivate() {}
    public void ejbActivate() {}

    public String sayHello() {
        String message = "Hello there!";
        try {
            Context initCtx = new InitialContext();
            Context beanEnv =
                (Context)initCtx.lookup("java:comp/env");
            message = (String)beanEnv.lookup("message");
        }
        catch(NamingException nex) {
            message = nex.getMessage();
        }
        return message;
    }

    public void setSessionContext(SessionContext ctx) {
        this.ctx = ctx;
    }
}

```

*Listing 3: Bean Implementation*

After compiling the above code the bean developer places the class files in a jar file. The jar file also includes a deployment descriptor file (see Appendix A) in XML format to provide bean information to the container. This file must contain bean structural and external dependency information and, optionally, application assembly information. Bean structural information includes a name for the bean; home, remote, and implementation class names; business method names, and environment properties. See EJB Specification 1.1 for full details. In this case the bean has a message property that can be easily changed after the bean is deployed. The bean deployer then uses container tools to deploy the bean in the container.

Now we need a client to access the bean. The client can be another bean, or an external program such as a Java application, applet, or server page (JSP). Regardless the client must obtain a reference to the bean using JNDI. Listing 4 shows a JSP that connects to the hello bean and invokes the `sayHello()` method to display a message.

```

<%@ page import= "edu.gasou.cs.hellobeans.*" %>
<%@ page import= "javax.naming.Context" %>
<%@ page import= "javax.naming.InitialContext" %>
<%@ page import= "javax.rmi.PortableRemoteObject" %>
<%@ page import= "javax.ejb.*" %>
<html>
<head>
<title>Hello World EJB</title>
</head>
<body>
<%
    InitialContext ic = new InitialContext();
    Object objRef = ic.lookup( "HelloWorldGary" );
    HelloHome home = (HelloHome)PortableRemoteObject.narrow(
objRef, HelloHome.class );
    Hello remote = home.create();
    out.println(remote.sayHello());
%>
</body>
</html>

```

*Listing 4: Hello Java Server Page*

Now let's develop a more complicated application needing more than one component and a database. This application allows a student to view their class grades on-line. The EJB beans will handle user login and retrieving grades from the database. The student will login using an HTML page and the student's grades will be displayed using a JSP that invokes bean methods and formats the grades. This application will also demonstrate delegation and using the container environment.

There are two beans GradesManager and Student. GradesManager is a session bean that hides the Student bean from the client. The client must make method calls on GradesManager, which then delegates the implementation to the Student session bean. In this case, the only method available is `getGrades()`. This method takes as arguments the class name, user name, and password and returns a `Vector` of information. The `Vector` includes the student's name and for each assignment the assignment name, maximum points, class average, and the student's grade on the assignment. Session beans were used for both components to increase performance and to limit the database operations to retrieving data. See Appendix B for complete code listings.

GradesManager has two methods `getGrades()` and `memberFactory()`. `getGrades()` uses `memberFactory()` to obtain a reference to the Student bean and then uses the Student bean `getGrades()` method to retrieve the grades. This is similar to obtaining a remote reference used by the JSP client in Listing 4, but the lookup refers to the container environment. The deployment descriptor must also have an EJB reference to associate the two beans as shown in Listing 5.

```
<ejb-ref>
  <ejb-ref-name>java:comp/env/ejb/Student</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>edu.gasou.cs.gradesbeans.StudentHome</home>
  <remote>edu.gasou.cs.gradesbeans.Student</remote>
  <ejb-link>Student</ejb-link>
</ejb-ref>
```

*Listing 5: GradesManager deployment descriptor reference to Student*

The Student bean obtains a student's grades from the database. It obtains a database connection by using a data source as shown in Listing 6. The data source is a name defined in the container, the container associates the name with the actual database. This allows the application deployer to change databases without changing the bean code. The reference to the data source is defined in the deployment descriptor using a resource reference as shown in Listing 7.

```
Context ctx = new InitialContext();
DataSource ds =
(DataSource)ctx.lookup("java:comp/env/jdbc/GradesDB");
Connection dbConnection = ds.getConnection();
```

*Listing 6: Using a data source to connect to a database*

```
<resource-ref>
  <res-ref-name>java:comp/env/jdbc/GradesDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

*Listing 7: Resource reference for the database*

---

## CONCLUSIONS

---

Component technologies make assembly and deployment of applications easier for the application assembler. The complexity is taken care of by the framework developer and the component developer. But, to make this possible standards must be established so that independently developed components will work together when assembled. These standards are evolving as component theory and application converge.

Enterprise JavaBeans is a component model that represents a point along this evolution. It is based on Java, an OO language, and does not fully support component concepts. Perhaps in the coming years as programmers become more familiar with the advantages of components and begin thinking in component design methodologies a true component language will emerge.

---

## REFERENCES

---

1. Bachman, Felix; et. al.: Technical Concepts of Component Based Engineering, Volume II, May 2000, Carnegie-Mellon Technical Report, ESC-TR-2000-007.
2. DeMichiel, Linda and Yalcinalp, Umit: *Enterprise JavaBeans Specification, Version 2.0 Proposed Final Draft*, Sun Microsystems, 2001.
3. D'Souza, Desmond and Wills, Alan: Objects, Components, and Frameworks with UML, Addison-Wesley, 1999.
4. Grand, Mark: Patterns in Java, Volume 1, Wiley & Sons, 1998.
5. Fayad, Mohamed and Schmidt, Douglas: *Object-Oriented Application Frameworks*, Communications of the ACM, Oct 1997, Vol 40 No 10.
6. Johnson, Ralph: *Frameworks=(Components+Patterns)*, Communications of the ACM, Oct 1997, Vol 40 No 10.
7. Matena, Vlada and Hapner, Mark: *Enterprise JavaBeans Specification, v1.1*, Sun Microsystems, 1999.
8. Monson-Haefel, Richard: Enterprise JavaBeans, O'Reilly, 1999.
9. Szyperski, Clemens: Component Software Beyond Object-Oriented Programming, Addison-Wesley, 1998.
10. Thomas, Anne: *Enterprise JavaBeans Technology*, Sun Microsystems, 1998.
11. Valesky, Tom: Enterprise JavaBeans Developing Component-Based Distributed Applications, Addison-Wesley, 1999.
12. Wang, Guijun; et. al.: *Component Assembly for OO Distributed Systems*, Computer, July 1999, Vol. 32, Issue 7.
13. [http://e-docs.beasys.com/wls/docs61/ejb/EJB\\_design.html#1029240](http://e-docs.beasys.com/wls/docs61/ejb/EJB_design.html#1029240), "WebLogic Server EJB Design and Development", BEA Systems, 2001.

---

## APPENDIX A: HELLO APPLICATION

---

### HELLO BEAN

#### REMOTE INTERFACE CODE

```
//Hello.java
package edu.gasou.cs.hellobeans;
import javax.ejb.*;
public interface Hello extends EJBObject
{
    String sayHello() throws java.rmi.RemoteException;
}
```

#### HOME INTERFACE CODE

```
//HelloHome.java
package edu.gasou.cs.hellobeans;
import javax.ejb.*;
public interface HelloHome extends EJBHome
{
    public Hello create() throws java.rmi.RemoteException,
        javax.ejb.CreateException;
}
```

#### BEAN CODE

```
//HelloBean.java
package edu.gasou.cs.hellobeans;
import javax.ejb.*;
import javax.naming.*;
public class HelloBean implements SessionBean
{
    SessionContext ctx;
    public void ejbCreate() {}
    public void ejbRemove() {}
    public void ejbPassivate() {}
    public void ejbActivate() {}

    public String sayHello() {
        String message = "Hello there!";
        try {
            Context initCtx = new InitialContext();
            Context beanEnv = (Context)initCtx.lookup("java:comp/env");
            message = (String)beanEnv.lookup("message");
        }
        catch(NamingException nex) {
            message = nex.getMessage();
        }
        return message;
    }

    public void setSessionContext(SessionContext ctx) {
        this.ctx = ctx;
    }
}
```

## DEPLOYMENT DESCRIPTOR

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 1.1//EN" 'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<ejb-jar>
<description></description>
<display-name>HelloWorldGary</display-name>
<enterprise-beans>
<session>
    <display-name>HelloWorldGary</display-name>
    <ejb-name>HelloWorldGary</ejb-name>
    <home>hellobeans.HelloHome</home>
    <remote>hellobeans.Hello</remote>
    <ejb-class>hellobeans.HelloBean</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
</session>
</enterprise-beans>
<assembly-descriptor>
    <container-transaction>
        <method>
            <ejb-name>HelloWorldGary</ejb-name>
            <method-intf>Remote</method-intf>
            <method-name>sayHello</method-name>
            <method-params />
        </method>
        <trans-attribute>Required</trans-attribute>
    </container-transaction>
</assembly-descriptor>
</ejb-jar>
```

## CLIENT JSP

```
<%@ page import= "hellobeans.*" %>
<%@ page import= "javax.naming.Context" %>
<%@ page import= "javax.naming.InitialContext" %>
<%@ page import= "javax.rmi.PortableRemoteObject" %>
<%@ page import= "javax.ejb.*" %>
<html>
<head>
<title>Hello World EJB</title>
</head>
<body>
<%
    InitialContext ic = new InitialContext();
    Object objRef = ic.lookup( "HelloWorldGary" );
    HelloHome home = (HelloHome)PortableRemoteObject.narrow( objRef,
HelloHome.class );
    Hello remote = home.create();
    out.println(remote.sayHello());
%>
</body>
</html>
```

---

## APPENDIX B: GRADES MANAGER

---

### GRADESMANAGER BEAN

#### REMOTE INTERFACE CODE

```
//GradesManager.java
package edu.gasou.cs.gradesbeans;
import javax.ejb.*;

public interface GradesManager extends EJBObject
{
    java.util.Vector getGrades(String user, String password, String
classname) throws java.rmi.RemoteException, java.lang.Exception,
        java.sql.SQLException;
}
```

#### HOME INTERFACE CODE

```
//GradesManagerHome.java
package edu.gasou.cs.gradesbeans;
import javax.ejb.*;

public interface GradesManagerHome extends EJBHome
{
    public GradesManager create() throws java.rmi.RemoteException,
        javax.ejb.CreateException;
}
```

#### BEAN CODE

```
//GradesManagerBean.java
package edu.gasou.cs.gradesbeans;

import javax.ejb.*;
import java.util.Vector;
//environment
import javax.naming.*;
import javax.rmi.PortableRemoteObject; //use RMI for remote references

public class GradesManagerBean implements SessionBean
{
    SessionContext ctx;

    public void ejbCreate() {}
    public void ejbRemove() {}
    public void ejbPassivate() {}
    public void ejbActivate() {}

    public void setSessionContext(SessionContext ctx) {
        this.ctx = ctx;
    }
}
```

```

/* Retrieve the grades for a student with user, pass, and class and
   return the grades in a Vector of Strings.
   Throws a SQLException if no user with password is found in the
   database.
*/
public Vector getGrades(String user, String password, String
classname) throws Exception, java.sql.SQLException
{
    //use factory to create a reference to a member
    Student mem = memberFactory();
    //get grades
    Vector grades = mem.getGrades(user, password,
classname);

    return grades;
}

//creates appropriate member
private Student memberFactory() throws Exception
{
    Student remote;
    //get a reference to a Student bean
    InitialContext ic = new InitialContext();
    Object objRef = ic.lookup( "java:comp/env/Student" );
    StudentHome home =
(StudentHome)PortableRemoteObject.narrow( objRef, StudentHome.class );
    remote = home.create();
    return remote;
} // end memberFactory
} //end GradesManagerBean

```

## **STUDENT BEAN**

### **REMOTE INTERFACE CODE**

```

//Student.java
package edu.gasou.cs.gradesbeans;
import javax.ejb.*;

public interface Student extends EJBObject
{
    java.util.Vector getGrades(String user,String password, String
classname) throws java.rmi.RemoteException, java.lang.Exception,
java.sql.SQLException;
}

```

## HOME INTERFACE CODE

```
//StudentHome.java
package edu.gasou.cs.gradesbeans;
import javax.ejb.*;

public interface StudentHome extends EJBHome
{
    public Student create() throws java.rmi.RemoteException,
                               javax.ejb.CreateException;
}
```

## BEAN CODE

```
//StudentBean.java
/* StudentBean.java

Retrieves the grades for a student from a database.
Throws an SQLException if the student doesn't exist or
the password is incorrect.

Places the student information in a Vector of String objects.
The data is arranged in the Vector as follows:

    student name
    *assignment name
    *points for assignment
    *class average for assignment
    *student grade for assignment
    *repeat for each assignment
*/

package edu.gasou.cs.gradesbeans;

import javax.ejb.*;
import javax.naming.*;
//jdbc imports
import java.sql.*;
import javax.sql.DataSource;
import java.util.Vector;
//formatting
import java.text.DecimalFormat;

public class StudentBean implements SessionBean
{
    public SessionContext ctx;

    public void ejbCreate() {}
    public void ejbRemove() {}
    public void ejbPassivate() {}
    public void ejbActivate() {}

    public void setSessionContext(SessionContext ctx) {
        this.ctx = ctx;
    }

/* Retrieve the grades for a student with user, password,
```

```

and return a Vector of data.
Throws exceptions. Client must catch and process the exceptions.
SQLException if user or password is incorrect.
*/
    public Vector getGrades(String user, String password, String
classname) throws Exception, SQLException
    {
        Vector grades;

        //Queries for an Access database
        //Retrieves grades for a student
        String sqlQuery = "SELECT * FROM " + classname + " WHERE
((Username='" + user + "') AND (Password='" + password + "'))";
        //Retrieves visible property for each assignment
        String sqlVisible = "SELECT * FROM " + classname + "
WHERE (Last='Visible')";
        //Retrieves maximum points for each assignment
        String sqlOutOf = "SELECT * FROM " + classname + " WHERE
(Last='OutOf')";
        //Retrieves the class average for each assignment
        String sqlAverages = "SELECT * FROM " + classname + "
WHERE (Last='Class')";

        /*Get a datasource so can connect to database.
The actual database connection is defined in the
container.
This gets a datasource reference to the connection
defined in the deployment descriptor.
*/
        Context ctx = new InitialContext();
        DataSource ds =
(DataSource)ctx.lookup("java:comp/env/jdbc/GradesDB");
        Connection dbConnection = ds.getConnection();

        //get resultset for student
        Statement stmtStudent = dbConnection.createStatement();
        ResultSet rsStudent =
stmtStudent.executeQuery(sqlQuery);

        //get resultset for the Visible row
        Statement stmtVisible = dbConnection.createStatement();
        ResultSet rsVisible =
stmtVisible.executeQuery(sqlVisible);

        //get resultset for the OutOf row
        Statement stmtOutOf = dbConnection.createStatement();
        ResultSet rsOutOf = stmtOutOf.executeQuery(sqlOutOf);

        //get resultset for the Class Average row
        Statement stmtAverages = dbConnection.createStatement();
        ResultSet rsAverages =
stmtAverages.executeQuery(sqlAverages);

        //extract the data from the result sets and place in a Vector

```

```

        grades = extractData(rsStudent, rsVisible, rsOutOf,
rsAverages, classname);

        //close recordsets
rsStudent.close();
rsVisible.close();
rsOutOf.close();

        //close statements
stmtStudent.close();
stmtVisible.close();
stmtOutOf.close();
        //close DB connection
dbConnection.close();

        return grades;
    }

    /* Extracts the data from the recordset and returns a Vector of
    Strings containing the students full name and grades for
    each assignment.
    */
    private Vector extractData(ResultSet rsStudent, ResultSet
rsVisible, ResultSet rsOutOf, ResultSet rsAverages, String classname)
throws Exception, SQLException
    {
        String assignment, grade=null, outof, average,
colTypeName;
        Vector grades = new Vector();
        Object objVisible = null;
        DecimalFormat twoDecimal = new DecimalFormat("0.0");

        //Get the number of columns of data in the recordsets
ResultSetMetaData rsmd = rsStudent.getMetaData();
int items = rsmd.getColumnCount();

        //go to the first record (should only be one record)
rsStudent.next();
rsVisible.next();
rsOutOf.next();
rsAverages.next();

        //get the student's first name and last name
        //first name is in column 2 and lastname is in column 1
grades.add( rsStudent.getString(2) + " " +
rsStudent.getString(1) );

        //get each assignment and grade
        // the first start-1 columns do not contain grades
int start = getStartCol();
for(int i=start; i<=items; i++)
    {

        /* Get the object in the Visible column; could be a
        number or a String.
        */
    }

```

```

objVisible = rsVisible.getObject(i);

/* Add the assignment information to the Vector if
there is a value in the Visible column*/
if(objVisible != null) {
    //get the assignment name
    assignment = rsmd.getColumnLabel(i);
    grades.add(assignment);
    //get the max points for the assignment
    outof = rsOutOf.getString(i);
    if(outof == null) outof = "";
    grades.add(outof);
    //get the class average for the assignment
    average = rsAverages.getString(i);
    grades.add(average);
    //get the grade for the assignment
    // could be a letter grade or a numeric grade
    colTypeName = rsmd.getColumnTypeName(i);
    colTypeName.trim();
    if(colTypeName.equals("VARCHAR"))
        grade = rsStudent.getString(i);
    else
        grade = twoDecimal.format(rsStudent.getDouble(i));
    if(grade == null) grade = "";
    grades.add(grade);
} // end if

} //end for loop

return grades;

} // end extractData()

//retrieve the starting column of the grades in the database
// stored in the container environment for the bean
private int getStartCol() throws Exception {
    Context ctx = new InitialContext();
    Integer start = (Integer)ctx.lookup("java:comp/env/startCol");
    return start.intValue();
}

} //end StudentBean

```

## **DEPLOYMENT DESCRIPTOR**

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 1.1//EN" 'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
<ejb-jar>
<description></description>
<display-name>GradesManager</display-name>
<enterprise-beans>
    <session>
        <ejb-name>GradesManager</ejb-name>

```

```

<home>edu.gasou.cs.gradesbeans.GradesManagerHome</home>
<remote>edu.gasou.cs.gradesbeans.GradesManager</remote>
<ejb-class>edu.gasou.cs.gradesbeans.GradesManagerBean</ejb-class>
<session-type>Stateless</session-type>
<transaction-type>Bean</transaction-type>
<ejb-ref>
  <ejb-ref-name>java:comp/env/ejb/Student</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>edu.gasou.cs.gradesbeans.StudentHome</home>
  <remote>edu.gasou.cs.gradesbeans.Student</remote>
  <ejb-link>Student</ejb-link>
</ejb-ref>
</session>
<session>
  <ejb-name>Student</ejb-name>
  <home>edu.gasou.cs.gradesbeans.StudentHome</home>
  <remote>edu.gasou.cs.gradesbeans.Student</remote>
  <ejb-class>edu.gasou.cs.gradesbeans.StudentBean</ejb-class>
  <session-type>Stateless</session-type>
  <transaction-type>Bean</transaction-type>
  <env-entry>
    <env-entry-name>startCol</env-entry-name>
    <env-entry-type>java.lang.Integer</env-entry-type>
    <env-entry-value>5</env-entry-value>
  </env-entry>
  <resource-ref>
    <res-ref-name>java:comp/env/jdbc/GradesDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
</session>
</enterprise-beans>
</ejb-jar>

```

## CLIENT HTML/JSP

```

<!GradesFormJSP.html>
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-
1252">
<META HTTP-EQUIV="Content-Language" CONTENT="en-us">
<TITLE>Grades Login</TITLE>
</HEAD>
<BODY background="images/background.jpg">
<H1>Grades Login for Mr. Huband</H1>
<HR>
<FORM METHOD="GET" ACTION="http://localhost:8100/GradesTable.jsp">
<BLOCKQUOTE>
<PRE><EM>      </EM><EM>      Class:      </EM><select size="1"
name="classname">
<option selected value="ProgPrinID">Programming Principles I D</option>
<option value="VBC">Visual BASIC C</option>
<option value="VBD">Visual BASIC D</option>
</select><em>

```

```

        User Name: </em><INPUT NAME="username" SIZE=25 MAXLENGTH=25>
<EM>        Password: </EM><INPUT TYPE=PASSWORD NAME="password"
SIZE=25 MAXLENGTH=25>
</PRE>
</BLOCKQUOTE>
<p align="center">
<INPUT TYPE=SUBMIT VALUE="Get Grades">
<INPUT TYPE=RESET VALUE="Clear Form">
</FORM>
<HR>
<H5>Copyright © 2001. All rights reserved. <BR>
Revised: <!--WEBBOT bot=TimeStamp
        S-Type="EDITED"
        S-Format="%B %d, %Y" startspan
-->August 25, 2001<!--WEBBOT bot=TimeStamp endspan i-checksum="21756"
--></H5>
</BODY>
</HTML>

```

```

<!GradesTable.jsp!>
  <%@ page import= "edu.gasou.cs.gradesbeans.*" %>
  <%@ page import= "javax.naming.Context" %>
  <%@ page import= "javax.naming.InitialContext" %>
  <%@ page import= "javax.rmi.PortableRemoteObject" %>
  <%@ page import= "javax.ejb.*" %>
  <%@ page import= "java.util.Vector" %>
  <%@ page import= "java.sql.SQLException" %>

<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-
1252">
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<title>Grades</title>
</head>

<body>

  <%
    //get the class, user name, and password from the login form
    String classname = request.getParameter("classname");
    String user = request.getParameter("username");
    String password = request.getParameter("password");
    //get a reference to the GradesManager bean
    try {
      InitialContext ic = new InitialContext();
      Object objRef = ic.lookup( "GradesManager" );

```

```

        GradesManagerHome home =
        (GradesManagerHome)PortableRemoteObject.narrow( objRef,
        GradesManagerHome.class );
        GradesManager remote = home.create();
        //get the vector of grades data
        Vector grades = remote.getGrades(user, password, classname);
        //display the class name and student name
        String studentname = (String)grades.get(0);
        out.println("<p><b>Class</b>: " + classname + "<br><b>Name</b>:
" + studentname + "</p>\n");
        %>

<table border="1" width="60%" cellspacing="1">
    <tr>
        <td width="15%"><b>Assignment</b></td>
        <td width="15%"><b>Assignment Points</b></td>
        <td width="15%"><b>Class Average</b></td>
        <td width="15%"><b>Grade</b></td>
    </tr>

    <%
        //put data for one assignment in a table row
        //variables to hold data
        String assignment;
        String points, average, grade;
        //generate assignment rows
        for(int i = 1; i < grades.size()-4; i+=4)
        {
            //extract data for one assignment
            assignment = (String)grades.get(i);
            points = (String)grades.get(i+1);
            average = (String)grades.get(i+2);
            grade = (String)grades.get(i+3);
            //put data into row columns
            out.println("<tr>\n"); //start table row
            out.println(" <td width='15%'>" + assignment +
"</td>\n"); //assignment column
            out.println(" <td width='15%'>" + points + "</td>\n");
            out.println(" <td width='15%'>" + average + "</td>\n");
            out.println(" <td width='15%'>" + grade + "</td>\n");
            out.println("</tr>\n"); //end table row
        }
    }catch(java.sql.SQLException ex) {
        out.println("<p><b>LOGIN ERROR! CLICK THE BACK BUTTON
AND TRY AGAIN.</b></p>");
    }
    %>
</table>
</body>
</html>

```